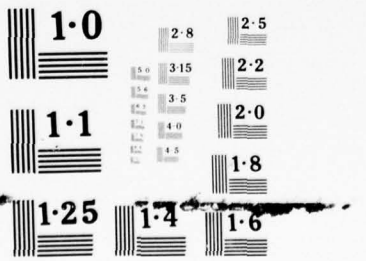
AD-A046 571 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY. SREP FINAL REPOR--ETC(U)
AUG 77 M W ALFORD, P H BROWNE, I F BURNS DAS60-75-C-0022
UNCLASSIFIED TRW-27332-6921-026-VOL-1 NL

1 OF 4
ADA
046571



1 OF 4
ADA
046571



NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

AD A046571

27332-6921-026

SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY

SREP FINAL REPORT - VOLUME I

CDRL C005

1 AUGUST 1977

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Prepared For
BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER
DASG60-75-C-0022

DDC
RECEIVED
NOV 18 1977
A

AD No.
DDC FILE COPY

TRW
DEFENSE AND SPACE SYSTEMS GROUP
HUNTSVILLE, ALABAMA

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CDRL C005 (Volume I)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Software Requirements Engineering Methodology, (SREP Final Report, Volume I)	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report
6. PERFORMING ORG. REPORT NUMBER: TRW-27332-6921-026-VOL-1	
7. AUTHOR(s): M. W. Alford, et al P. H. Browne, I. F. Burns, H. A. Helton G. C. Hitt	8. CONTRACT OR GRANT NUMBER(s): DASG60-75-C-0022
9. PERFORMING ORGANIZATION NAME AND ADDRESS: TRW Defense and Space Systems Group, 7702 Governors Drive, West, Huntsville, Alabama 35805	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 6.33.04.A
11. CONTROLLING OFFICE NAME AND ADDRESS: Ballistic Missile Defense Advanced Technology Center, P. O. Box 1500, Huntsville, AL 35807, ATTN: ATC-P	12. REPORT DATE: 1 August 1977
	13. NUMBER OF PAGES: 378
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office): (12) 380p.	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Cleared for public release - distribution unlimited.
Reference BMDSC-CRS letter dated 8 March 1977.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software
Software Specification
Software Requirements
Software Development

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A methodology is presented for the development and management of software specifications. The technique is built upon a language (RSL) readable both by a computer and by man, and a set of tools termed collectively the Requirements Engineering and Validation System (REVS). The tools provided for retention of all requirements in a relational data base from which documentation, consistency analyses, and simulations may be constructed automatically.

over →

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

407674

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (Continued)

The methodology systematically develops the specification from source documentation at the system level, documenting omissions and errors of the source materials in the process. The produced requirements are provably consistent, and may be validated against system objectives through the generated simulation. The entire process is subject to systematic management through definable and verifiable milestones supported by REVS.



27332-6921-026

SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY

SREP FINAL REPORT - VOLUME I

CDRL C005

1 AUGUST 1977

CLEARED FOR PUBLIC RELEASE - DISTRIBUTION UNLIMITED. REFERENCE BMDSC-CRS LETTER DATED 8 MARCH 1977.

THE FINDINGS OF THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION.

Prepared For

BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER

DASG60-75-C-0022

TRW

DEFENSE AND SPACE SYSTEMS GROUP
Huntsville, Alabama

27332-6921-026

SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY

SREP FINAL REPORT - VOLUME I

CDRL C005

1 AUGUST 1977

Principal Authors:

M. W. Alford
P. H. Browne
I. F. Burns
H. A. Helton
G. C. Hitt
J. T. Lawson
M. D. Richter
R. J. Smith

Approved By:

L. R. Marker, Manager
Software Requirements
Engineering Program

P. H. Browne, Head
Methodology Research
and Development

James E. Long, Manager
Huntsville Facility

Prepared For

BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER

DASG60-75-C-0022

TRW
DEFENSE AND SPACE SYSTEMS GROUP

Huntsville, Alabama

TABLE OF CONTENTS

Section	Title	Page
1.0	INTRODUCTION	1-1
	1.1 SCOPE AND CONTENT	1-3
	1.2 THE PLANNED ROLE OF SREM	1-4
	1.3 CONSIDERATIONS OF SREM DEVELOPMENT	1-6
	1.4 MAJOR COMPONENTS OF SREM	1-8
	1.4.1 The Requirements Statement Language (RSL)	1-8
	1.4.2 The Requirements Engineering and Validation System (REVS)	1-8
	1.4.3 The Engineering Methodology	1-10
	1.4.4 Specification Management	1-11
	1.5 APPLICABILITY	1-12
	1.6 TERMINOLOGY	1-12
	PART I - TECHNICAL APPROACH	
2.0	SREM OVERVIEW	2-1
	2.1 THE TRACK LOOP SYSTEM EXAMPLE	2-3
	2.1.1 Preliminary Ballistic Missile Defense System	2-5
	2.1.2 TLS Requirements	2-5
	2.2 SUMMARY OF APPENDICES	2-5
3.0	FUNCTIONAL REQUIREMENTS	3-1
	3.1 PHASE 1 - DEFINITION OF SUBSYSTEM ELEMEMTS	3-4
	3.1.1 Initial Inputs	3-5
	3.1.2 Interface Definition	3-7
	3.1.3 Message Definition	3-8
	3.1.4 The Interface Data Hierarchy	3-15
	3.1.5 Problems of Definition	3-17
	3.1.6 R_NET Definition	3-19
	3.1.7 Entity Definition	3-26
	3.1.8 The Entity Data Hierarchy	3-28
	3.1.9 Independent FILEs	3-31
	3.1.10 Summary of Phase 1	3-32
	3.2 PHASE 2 - EVALUATION OF THE KERNEL	3-34
	3.2.1 Data Naming Conventions	3-34
	3.2.2 Structural Data Definitions	3-35
	3.2.3 Entering R_NETs in the ASSM	3-37
	3.2.4 The STRUCTURE of an R_NET	3-37

TABLE OF CONTENTS (Continued)

Section	Title	Page
	3.2.5 Checking the Kernel with the Aid of RADX . . .	3-38
	3.2.6 Summary of Phase 2	3-40
	3.3 PHASE 3 - COMPLETION OF THE FUNCTIONAL DEFINITION . .	3-42
	3.3.1 Entity Transitions	3-43
	3.3.2 Data Transactions.	3-44
	3.3.3 RADX Evaluation of Data Transactions	3-46
	3.3.4 Hierarchy Transitions.	3-48
	3.3.5 Further Data Definition.	3-49
	3.3.5.1 Locality.	3-50
	3.3.5.2 Type and Range.	3-51
	3.3.5.3 Use	3-53
	3.3.5.4 Values.	3-53
	3.3.6 Evaluation of the ASSM Using RADX.	3-54
	3.3.6.1 RADX Evaluation of Data Origin and Usage	3-55
	3.3.6.2 RADX Evaluation of File Activity. . .	3-56
	3.3.6.3 RADX Evaluation of Entity Activity. .	3-57
	3.3.6.4 RADX Evaluation of Data Attributes. .	3-58
	3.3.6.5 RADX Data Flow Analysis	3-60
	3.3.7 Summary of Phase 3	3-61
	3.4 PHASE 4 - COMPLETION OF MANAGEMENT AND CONTROL INFORMATION	3-62
	3.4.1 Originating Requirements	3-62
	3.4.2 Sources.	3-62
	3.4.3 Decisions.	3-63
	3.4.4 Relating to Sources.	3-64
	3.4.5 Informative Material	3-64
	3.4.5.1 Description	3-65
	3.4.5.2 Synonym	3-65
	3.4.5.3 Authorship.	3-65
	3.4.5.4 Complementary Relationships	3-66
	3.4.5.5 Structural References	3-66
	3.5 PHASE 5 - DYNAMIC FUNCTIONAL VALIDATION	3-67
	3.5.1 Betas.	3-67
	3.5.2 Local Data	3-68
	3.5.3 SETS Development	3-69
	3.5.4 Simulator Applications	3-70
4.0	PERFORMANCE REQUIREMENTS.	4-1
	4.1 PHASE 6 - ORIGINATING_REQUIREMENT DECOMPOSITION . . .	4-3

TABLE OF CONTENTS (Continued)

Section	Title	Page
	4.1.1 Step 1: ORIGINATING_REQUIREMENT Identification	4-3
	4.1.2 Step 2: Preliminary PERFORMANCE_REQUIREMENT Definition	4-3
	4.1.2.1 Decomposition of ORIGINATING_ REQUIREMENTs	4-7
	4.1.2.2 Preliminary Definition of Performance Requirement TESTs	4-9
	4.1.2.3 Definition of VALIDATION_POINTs and VALIDATION_PATHs	4-9
	4.2 PHASE 7 - PERFORMANCE ALLOCATION	4-17
	4.3 PHASE 8 - ANALYTIC FEASIBILITY DEMONSTRATION	4-20
	4.3.1 Form of Feasibility Demonstration	4-21
	4.3.2 GAMMA Development	4-21
	4.3.3 Benefits Derived	4-22
	PART II - MANAGEMENT APPROACH	
5.0	INTRODUCTION	5-1
6.0	DEFINING MEASURABLE MILESTONES	6-1
	6.1 TECHNICAL MILESTONES	6-3
	6.2 REVIEW AND CONTROL MILESTONES	6-5
	6.3 SUMMARY	6-8
7.0	COST AND SXHEDULE PLANNING	7-1
	7.1 EXPERIENCE TO DATE	7-2
	7.2 COST MODEL FORMULATION	7-4
	7.2.1 Cost Model 1 (CM1)	7-4
	7.2.1.1 Basic Cost Units	7-5
	7.2.1.2 Intangible Factors	7-8
	7.2.1.3 Cost of Supporting Elements	7-11
	7.2.2 Cost Model 2 (CM2)	7-13
8.0	MANAGEMENT CONTROL	8-1
	8.1 CONTROL MECHANISMS	8-1
	8.2 CHANGE CONTROL	8-4
	8.3 ACCEPTANCE OF THE SOFTWARE REQUIREMENTS	8-7
9.0	CONCLUSIONS	9-1

TABLE OF CONTENTS (Continued)

Section	Title	Page
10.0	APPLICABLE DOCUMENTS	10-1
APPENDIX A -	SAMPLE DPSPR - TRACK LOOP EXPERIMENT.	A-1
APPENDIX B -	RADAR/DPS INTERFACE SPECIFICATION	B-1
APPENDIX C -	C^2/DPS INTERFACE SPECIFICATION.	C-1
APPENDIX D -	RSL TERMINOLOGY	D-1
APPENDIX E -	RSL SUMMARY OF ELEMENT-TYPE	E-1
APPENDIX F -	TRACK LOOP SYSTEM RSL REQUIREMENTS KERNEL	F-1
APPENDIX G -	TLS REQUIREMENTS NETWORKS	G-1
APPENDIX H -	TLS REQUIREMENTS.	H-1

LIST OF ILLUSTRATIONS

Figure	Title	Page
1-1	The Cost of Delayed Error Detection	1-2
2-1	Track Loop System	2-4
3-1	RSL Subsystem Entries	3-6
3-2	An Elementary Interface Hierarchy	3-9
3-3	C^2 Input Hierarchy	3-13
3-4	RSL Message Entries	3-14
3-5	Interface Data Hierarchy	3-16
3-6	RSL Decision Entry	3-18
3-7	Three Asynchronous R_NETs	3-24
3-8	Entity Data Hierarchy	3-27
3-9	Entity Hierarchy	3-30
3-10	RSL Entity Entry	3-31
3-11	RSL Data Entry	3-36
3-12	RSL Initial ALPHA Entry	3-46
3-13	RSL Additional ALPHA Entry	3-49
4-1	Sample ORIGINATING_REQUIREMENTs (Performance) for TLS	4-4
4-2	RSL Representation of Performance Requirements at End of Phase 6 (TSL Example)	4-13
6-1	Overview of SREM Activities (Development and Validation of Functional Requirements)	6-2
6-2	Sample (Technical) Activity Network for SREM Phase 1	6-4
6-3	Complete Sample Activity Network for SREM Phase 1	6-6
6-4	Sample Decisions from Track Loop	6-7
7-1	Primary RSL Elements Affecting Basic Cost (Functional Requirements)	7-6
7-2	Basic Cost Unit Relationships	7-7
7-3	Predicted Impact Relationship	7-12
7-4	Cost Summed Over Major Elements	7-15

LIST OF ILLUSTRATIONS (Continued)

Figure	Title	Page
8-1	Interests Shared by SRE	8-2
8-2	Sample Change Flow into Update of Baseline.	8-6

LIST OF TABLES

Table No.	Title	Page
7.1	Resources Expended on Completed SREM Applications	7-3
7.2	Elements Generated Per Project.	7-9
7.3	Intangible Factors.	7-10
7.4	Rating of Intangible Factors.	7-12
7.5	Percentage Cost Breakdown	7-16
8.1	SREM Focus of Management Control on Substantive Issues. . .	8-3
8.2	Control Mechanisms for Major Managerial Control Issues. . .	8-5
8.3	Acceptance of the Software Requirements	8-8

1.0 INTRODUCTION

In the Fall of 1974 the Data Processing Directorate of the Ballistic Missile Defense Advanced Technology Center (BMDATC) initiated a series of research programs directed to the development of a complete and unified approach to software development. These programs encompassed the total range of activities from development of system specifications through completion and testing of the software process design. Supporting programs were also conducted to perform basic research into such areas as software reliability, static and dynamic validation techniques, and adaptive control and learning.

A key element of the BMDATC programs was the Software Requirements Engineering Program (SREP) -- a research effort concerned with a systematic approach to the development of complete and validated software requirements specifications. The overall objectives of this research were to:

- Ensure a well-defined technique for decomposition of system requirements into structured software requirements.
- Provide a mechanism for enhanced management visibility into the requirements development.
- Maintain requirements development independent of the target machine and the eventual software design.
- Allow for easy response to system requirement changes.
- Provide for testable and easily validated software requirements.

To meet these objectives, the Software Requirements Engineering Methodology (SREM) was developed. This methodology is an integrated, structured approach to requirements engineering activities. SREM begins with the translation and decomposition of system level requirements; performs analysis, definition, and validation of the software requirements; and ends with documentation of the software requirements in a Process Performance Requirements (PPR) specification. As part of the SRE methodology, a set of software support tools were implemented to automate many of the previously manual activities associated with requirements engineering. These software tools form the Requirements Engineering and Validation System (REVS); REVS processing is accomplished by expression of the software requirements in the structured, formal Requirements Specification Language (RSL).

SREM represents a different approach and philosophy for software requirements engineering. It embodies a flow orientation that eliminates many of the problems inherent in the classical functional hierarchy. It expresses requirements in an unambiguous, machine-processable language as opposed to free form (and free content) English. Support software is available to automatically process the requirements statements and perform a wide range of needed activities (e.g., static flow analysis, specification generation, simulation generation).

SREM is a comprehensive approach that was developed specifically to improve software requirements engineering practices. As shown in Figure 1-1, errors made in the requirements phase become increasingly more expensive to locate and correct in the later phases of development.

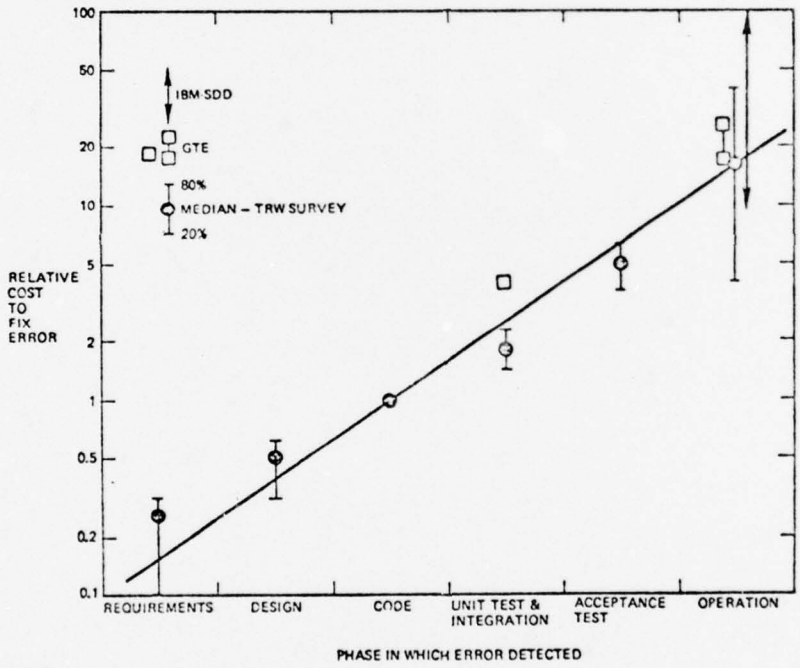


Figure 1-1 The Cost of Delayed Error Detection

Ambiguity and lack of preciseness in the requirements statements which indirectly promote misinterpretation (and therefore errors) in the subsequent phases also add to the cost and schedule leverage.

1.1 SCOPE AND CONTENT

This report is a SREM User's Guide for development of software requirements specifications. It is not a cookbook, in that it does not attempt the inherently impossible task of converting the genuinely creative aspects of specification development into rote, deductive operations. It does, however, define guidelines through which these creative operations are recognized, applied and restricted to their natural roles in the specification development process. In this manner, the scale and range of creativity can be defined and contained, thus allowing the specification development process to be scheduled with some degree of confidence. It should be noted that creative features remain in the methodology and as a consequence the major development effort must be conducted by experienced, knowledgable engineers. However, SREM has been structured in a manner such that many elements of specification development can be identified and isolated to permit less senior engineering personnel to perform the details of specification statement definition and documentation preparation.

This volume is organized in two major parts: Part I addresses the technical aspects of software requirements engineering. The methodology is described in detail, step-by-step, in the context of a representative example. Part II discusses the management of the specification development process with emphasis on how the specific features of SREM and its tools can be used to advantage in the management of the activities.

The remainder of this section describes the basic role of SREM, its major components and its position within the overall BMDATC software development methodology.

1.2 THE PLANNED ROLE OF SREM

SREM was conceived of, and designed as, an integral part of the BMDATC Software Development Methodology. The objective of the overall BMDATC methodology was to further the state-of-the-art in the specification, design, implementation and test of real-time BMD software. Its application was targeted to systems whose major characteristics and development environments are as follows:

System Characteristics

- Systems are large to very large
- Time responses are critical
- Processing is intensive
- Data base is large but not massive
- Computer system is digital with memory
- Technology of the object system is not fully understood initially (i.e., existence and feasibility of the system and subsystems is an issue)
- Requirements may be subject to a high degree of change
- Subsystems interfacing with the DP may (probably) be undergoing parallel development (i.e., interfaces are flexible).

Range of System Development Environments

- From a deployable system being developed essentially top-down from system performance objectives, which must deal with:
  - Hard performance requirements,
  - Firm threat definition,
  - Strong top-level configuration control,
  - Slow change control,
  - Maximum design freedom, and
  - Cost and schedule as major considerations.

- To an R&D project where design decisions are influenced by objectives of investigating specific approaches (e.g., improvement of the state-of-the-art) which must deal with
  - Minimum performance requirements,
  - Flexible threat,
  - Less-formal configuration control,
  - Reduced design freedom, and
  - Cost and schedule of lesser importance.

The range of activities which are addressed by SREM requires both a knowledge of systems engineering and of data processing technology. The starting point of SREM is the point in systems engineering when the system requirements analysis has identified the functions and the stress points of the weapon system; the interfaces between the subsystems (at least on the functional level); top-level weapon system functions and the weapon system operating rules (conditional statements impacting when and in what sequence the functions are performed); and the top-level weapon system functions that have been allocated to the data processor.

The ending point of SREM is reached when all system requirements allocated to the data processor have been sufficiently decomposed into data processor terms so that primarily software development expertise is required to continue. The DP interfaces have been determined to the element level, all processing steps have been identified with appropriate DP requirements levied, all actions of the DP in response to a stimulus are determined in terms of sequences of processing steps, and the processing necessary to generate all required DP output interface messages has been specified.

1.3 CONSIDERATIONS OF SREM DEVELOPMENT

The first step in defining the Software Requirements Engineering Methodology was to determine the properties required of a specification and of the individual requirements of which it is composed. The initial considerations were that:

- A specification is the set of all requirements which must be satisfied, and the identification of the subsets which must be met concurrently; and
- A specification is neither realizable nor legally binding unless it is consistent with both the laws of logic and the laws of nature.

In addition

- A specification defines the properties required of a product such that any delivery satisfying the specification satisfies the objectives of the specifier.

Taken together, the above consideration lead to a set of properties which a specification must have from a technical point of view.

- Internal Consistency
- Consistency with the physical universe
- Freedom from ambiguity.

Economic and management considerations lead to an additional set of properties which a good specification must exhibit:

- Clarity
- Minimality
- Predictability of specification development
- Controllability of software development.

Since freedom from ambiguity was mandatory, a machine-readable statement of software requirements was defined. It is a known principle of computer operation that input ambiguities can be tolerated only insofar as they are designed into the software. Thus, by employing an unambiguous language, and by translating and analyzing it with a program intolerant of ambiguity, a precise statement of requirements was ensured.

To provide an internally consistent specification, analyses of the requirements statements are incorporated into the system supporting the language. These analyses include semantic and syntactic decomposition of the individual statements, and analysis of the composite flow of data and processing. Support of consistency with the physical universe is accomplished by converting the specification unambiguously into a model (simulation) which can be executed against a model of the real world. Recently, the government has required that software be developed in accordance with a new standard, 5000.29. One key aspect of the new requirement is that any software specification must be validated before being imposed. With the collection of tools and the methodology for their use, SREM provides a means for that validation through static and dynamic analysis at the requirements level.

Finally, to support control of both the specification process and software development, a means of selective documentation and analysis of the specification is provided. The integration of these tools with a sound and methodical engineering and management approach provides predictability in the specification process and aids in avoiding overspecification.

1.4 MAJOR COMPONENTS OF SREM

1.4.1 The Requirements Statement Language (RSL)

The Requirements Statement Language is a user-oriented mechanism for specifying requirements. It is oriented heavily toward colloquial English, and uses nouns for elements and attributes, and transitive verbs for relationships; a complementary relationship uses the passive form of the verb. Both syntax and semantics echo English usage, so that many simple RSL sentences may be read as English with the same meaning. However, the precision of RSL, enforced through machine translation, is not typical of colloquial speech; as a result, most complex RSL sentences are a somewhat stylized form of English.

RSL is an extensible language in that certain primitive concepts are initially built in which can then, in turn, be used to define additional complex language concepts. The primitives are elements, attributes, relationships, and structures. From these, a nucleus of concepts has been defined which, to date, has proven sufficient. Future users of the language can add to the nucleus by means of the extension features provided by REVS.

1.4.2 The Requirements Engineering and Validation System (REVS)

REVS is an integrated set of tools used to support the definition, analysis, simulation, and documentation of software requirements. A key concept of REVS is that all requirements are translated into a central data base called the Abstract System Semantic Model (ASSM). The RSL statements themselves are not stored in the ASSM. Instead, they are translated into representations of the information content of the requirements statements. This provides an efficient and flexible means of maintaining a large software specification in a relatively small computer data base.

The ASSM is a relational data base providing a common source for all requirements analysis, modelling and for documentation. The commonality of all data ensures that any combination of extractions from the ASSM at any time (e.g., a document and a simulation) will be mutually consistent. That consistency is essential to asserting that the requirements modelled in validation of the specification are equivalent in every sense to those written in the specification.

REVS provides two mechanisms for entry of data into the ASSM, translation and interactive graphics, and a powerful set of tools for analysis termed collectively Requirements Analysis and Data Extraction (RADX). Translation is the process of converting RSL statements into the ASSM information, where the source of the statements may be cards, card images on tape, or keyboard entry from a terminal. Interactive graphics (RNETGEN) is a software package executing in conjunction with the Anagraph color graphics console to provide ASSM entry and illustrative documentation. It permits entry of structures and referenced elements in a manner parallel with the translator, and in fact may be used in conjunction with translation in an operational environment. Significantly, RNETGEN allows the user to attribute graphical information to his structure, both for multicolor display on the Anagraph and for documentation via CALCOMP.

Information held in the ASSM may be selected and output using RADX. That tool is responsive to user direction in selecting either a re-creation of the information translated into the ASSM, or the formatted abstraction of that information in a user-defined HIERARCHY. The combination of these features allows complex selections to be effected, so that all information needed for documentation and much that is essential to configuration management can be abstracted from the system without the encumberance of irrelevant data. Since all data abstractions are drawn from a common ASSM (and since that data base is confirmably consistent within itself), even redundant assertions in data extractions are absolutely consistent with one another.

Both static and dynamic analysis are provided by REVS in order to determine the internal consistency of the ASSM and its validity with respect to the laws of nature. Static analysis is performed in RADX which examines the data connectivity through the requirements to determine

that the laws of logic and the conventions of the language are fully satisfied throughout. Some forms of completeness testing are also accomplished, determining, for example, that constants are provided as required; the scope of completeness testing is largely at the discretion of the user, since he may define extensive static analyses through RADX commands to supplement those inherent in the system.

No amount of static testing can fully validate a set of requirements. To do so, the system they represent must be exercised against a model of the environment in which the system is to execute. Such simulations are provided by an automated simulation builder (SIMGEN), and a software package supporting its execution (SIMXQT). Two different levels of simulation are supported: analytic, in which high-fidelity models of the environment and explicit performance measures are provided, and functional, in which the connectivity of the system is validated with non-analytic models.

1.4.3 The Engineering Methodology

Historically, the methods for developing a software specification have been as numerous as the developers of such documents. In fact, few cases can be cited in which any formal methodology could be quoted. Until the specification appeared (often after tens or hundreds of man-years of effort), nothing was available to show that it would actually be generated. In addition, it has frequently been true that the quality of the specification even with respect to elementary consistency from one requirement to another, could be verified only very late in software development. Since the problems were discovered only when the cost of correction was prohibitive, the requirements were frequently changed, degrading system performance in order to have a 'workable' product.

The methodology developed within SREM is not only formal, in that it provides an explicit sequence of steps leading to the specification, but also manageable, in that it illuminates multiple phases for management review and analysis. Along the way, it supports early detection of high-level anomalies, since it works from the highest levels of software definition (processing and data flows) to the most detailed (analytic models and data content) in a systematic manner. A key feature of SREM is that the processing functions and data communications are considered in parallel,

rather than have either follow the other. As a result, the connectivity of the system is always complete, and it becomes possible to partition the requirements effort among several groups early in the process without risking divergence, omissions, or inconsistencies.

1.4.4 Specification Management

The management of a specification developed under SREM benefits most from the common source in the ASSM for all representations of the requirements. Thus, the simulation of the specification and the documentation of its requirements must be consistent at any time, since both have a single source of data which generate each without human intervention. In addition to a common data base, the methodology itself supports an orderly development which can be annotated with milestones, recorded on PERT charts, and otherwise controlled with the tools of the last several decades to provide predictability and control. This is not to suggest that the creativity of the specification process can either be scheduled or bypassed; it is still needed, but the methodology isolates it into segments with high visibility, supporting management cognizance of its progress and impact.

1.5 APPLICABILITY

These tools and techniques have been developed to address the needs of BMD software development. With perhaps minor exceptions, however, SREM is directly applicable to the specification of the requirements for any central software process for a large real-time system. In fact, since the methodology is inherently and deliberately computer independent, the techniques are not limited strictly to software in the form of computer programs. The requirements for any process composed of logical decisions and computations performed on data can be expressed via SREM -- regardless of whether the end product will be software, hardware, firmware, or some combination.

1.6 TERMINOLOGY

At the risk of introducing confusion, some non-standard terminology is used to describe certain SREM concepts. This has been done for two purposes: (1) to emphasize the different interpretations given to selected concepts, and (2) to emphasize the generality of the methodology application. An example of the first is the use of the term ALPHA for a processing step. The more common term "function" would be misleading because there is, in fact, a wide variety of common interpretations of "function". To avoid misunderstanding, a new, unfamiliar term is used in order to emphasize its specific meaning. An example of the second is the name applied to the resulting requirements specification which is called the Process Performance Requirements (PPR). No documentation system currently recognizes a document called a PPR. Here, the point is that any software requirements specification, whether called a B-5 (in MIL-STD 490) or something else in some other system, must contain a certain set of information. That specific set of information generated via SREM has been named a PPR. The same considerations lead to a definition of the originating input specifications to SREM as the Data Processing System Performance Requirements (DPSPR).

PART I - TECHNICAL APPROACH

2.0 SREM OVERVIEW

The Software Requirements Engineering Methodology has been developed in parallel with development of the Requirements Engineering and Validation System. As a consequence of this integrated development, there is a clean and comprehensible compatibility between the methodology and the instruments it uses to formulate and test a requirements specification. While REVS embodies the language and tools required for orderly development of process requirements specifications, SREM defines the techniques and procedures within which the tools and sound engineering and management practices are combined to generate a specification containing the desired properties.

SREM encompasses four major areas of engineering activity that begin with receipt of the set of information which defines the system level requirements on the Data Processing Subsystem. This input set is denoted as the Data Processing System Performance Requirements (DPSPR) Specification. The DPSPR includes the Data Processing System Interface Requirements Specifications and any external subsystem Performance Requirements Specifications which influence the definition of the Process Performance Requirements. Using these source documents as a stimulus, the requirements engineer becomes involved in the four major engineering activities defined by SREM to develop the Process Performance Requirements Specification. These engineering activities are:

- Identification, definition and development of the functional requirements,
- Identification, definition and development of the performance requirements,
- Development of the Process Performance Requirements Specification and
- Development of the analytic feasibility demonstrations.

The inherently sequential nature of the steps of the methodology appeared at first to make incremental specification of software awkward. Experience on many programs, notably Systems Technology Program*, has made it clear that any new technology should assume that knowledge of requirements will increase continuously throughout the development of the software specification, rather than be complete when software requirements are first initiated. Thus, the tools and methodology of SREP were developed to allow for incremental development of the specification. Specific features, such as VERSION and the qualified inclusion of R_NETs in a simulation provide the capability either for defining segments of the software requirements, or for augmenting a full subsystem with additional functions. The consistency and integrity enforced by the system are fundamental to success in incremental specification. They ensure that:

- Portions of the system specified later than some segments will be consistent since their connectivity with the early segments was defined at the highest levels; and
- Any extension of the system will be compatible with prior specifications, since any inconsistency would preclude entering the extension into the ASSM.

During each activity of SREM the features of REVS are utilized to control, monitor, test and maintain the evolving collection of requirements statements. The functional requirements are defined in RSL statements and catalogued by REVS in the ASSM by the RSL Translator. The accuracy and correctness of these RSL statements is verified by the Statis Analyzer portion of the RADX segment of REVS. Continuity and completeness of these RSL statements is analyzed through the Simulator Generation and Simulation Execution functions of REVS using algorithms for each functional requirement represented as executable PASCAL procedures implemented as BETA models. Next, the performance requirements are defined in RSL statements and catalogued by REVS in the ASSM (by the RSL Translator) and attached to the functional requirements each CONTRAINTS. The accuracy and correctness of these RSL statements are again verified by the Static Analyzer portion of

*Formerly the Site Defense Program

the RADX segment of REVS. Continuity and completeness of these RSL statements is analyzed through the Simulator Generator and Simulation Execution functions of REVS using algorithms for each functional requirement, represented as executable PASCAL procedures implemented as GAMMA models. Validation of the functional and performance requirements testability is confirmed by REVS through execution of TESTs implemented as executable PASCAL procedures. In this way, the existence of a feasible design solution for the collection of functional and performance requirements statements is confirmed by REVS through candidate algorithms used as the GAMMA models, and a model of the System Environment and Threat Simulation (SETS). The models are executed against one another with a variety of scenarios to demonstrate the existence of a solution to the requirements statement in the ASSM. Finally, data collected through RADX are formatted and published as a Process Performance Requirements Specification.

The detailed description of the SREM technique of specifying software functional and performance requirements is presented in Sections 3 and 4. The methodology is described in the context of an example which is defined to the degree necessary to illustrate the method. The example is a hypothetical system called Track Loop System (TLS). TLS is representative of the kind and complexity of real BMD systems, and yet is simple enough to serve as a comprehensible example. A complete DPSPR (including the interface specifications) for TLS is provided as Appendix A. The system is summarized below.

2.1 THE TRACK LOOP SYSTEM EXAMPLE

The Track Loop System (TLS) is a subset of a Preliminary Ballistic Missile Defense System that is capable of nearly autonomous execution in response to external stimuli. It is the simplest known BMD subsystem with properties of interest for software definition, and it is one which has been studied extensively, both in the academic literature and in such practical programs as Site Defense. Therefore, it has been selected as the testbed for supporting experimentation in development of the methodology for software requirements. A pictorial representation of the TLS is provided in Figure 2-1.

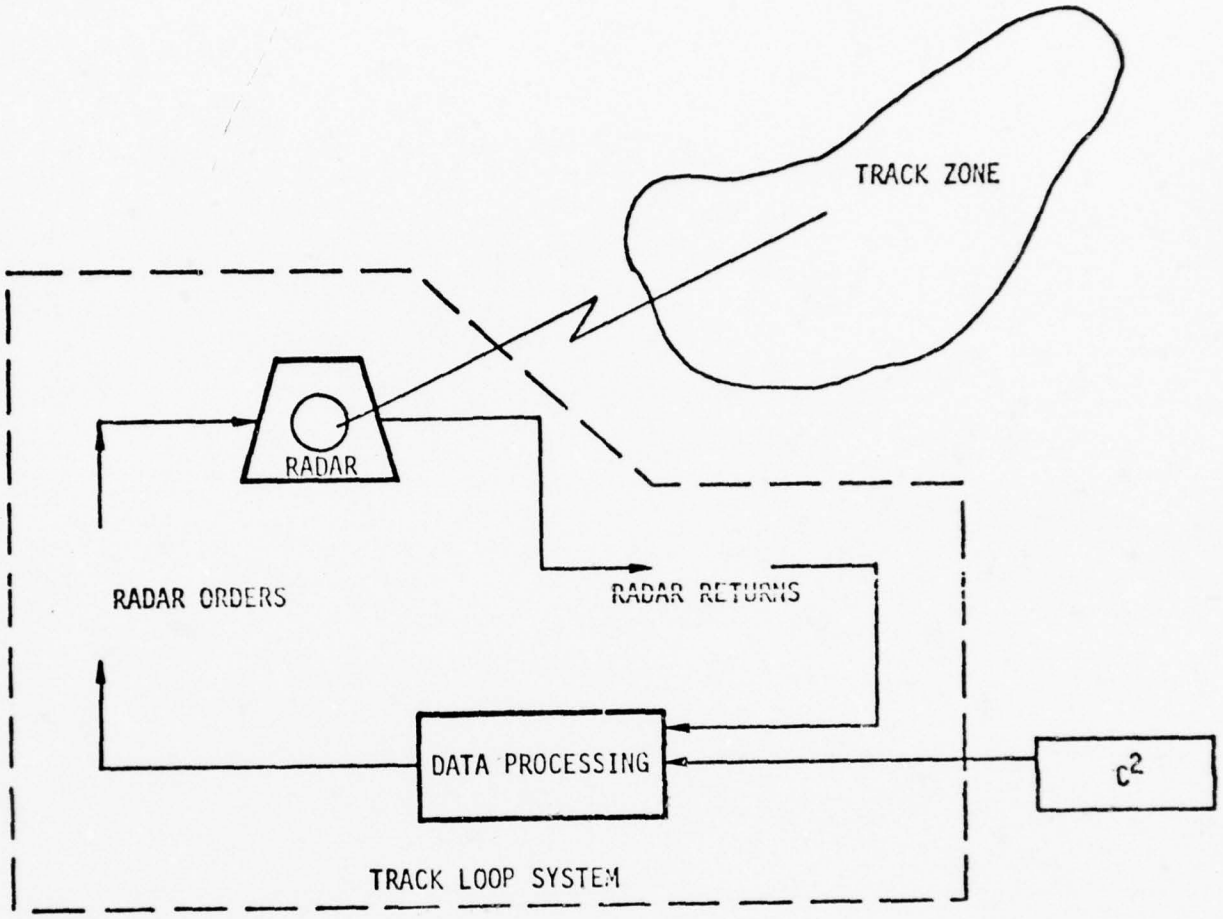


Figure 2-1 Track Loop System

2.1.1 Preliminary Ballistic Missile Defense System

A Preliminary Ballistic Missile Defense System (PBMDS) has been postulated as an environment in which the TLS would execute. It is a generalized representative of the class of systems currently in development, and is particularized for the TLS through representative but non-real specifications where required. In the Conduct Engagement mode, an object entering the search region will be detected and designated, tracked, discriminated, and engaged (as required) in defense of the ground facilities. Those functions are implemented through the Data Processing System (DPS), a radar or other sensor, and a means of neutralizing hostile objects. For the purpose of the TLS, only the radar need be defined in detail; other system elements are identified only to the extent that they impact DPS requirements.

2.1.2 TLS Requirements

The TLS is required to perform five system level functions: 1) system initialization and engagement initiation, 2) engagement termination, 3) target tracking, 4) control of system resources, and 5) recording of data during the engagement. The system includes: the DPS, the radar and the recording media, and directly interfaces with the external environment through communications and control (C^2).

In general, the functions of TLS are initiated by messages from C^2; however, track maintenance and certain control functions are autonomous. The engagement is initiated and terminated by C^2 messages; during engagement, radar data are reported to the DP periodically. When an image is handed over to TLS through C^2, it is tracked without further direction, until it is dropped either by command or by determination within the DPS. This configuration thus demonstrates both exogenous and endogenous process excitation, and in other ways provides a microcosm of a BMD process.

2.2 SUMMARY OF APPENDICES

The Appendices provide a summary of the Requirements Statement Language and a complete development of the TLS requirements statements. A complete description of RSL is provided in the REVS Users Manual (Volume II of this report).

Appendices A through C contain the TLS source specifications from which the TLS requirements were developed. Appendix D summarizes the RSL Terminology by providing a copy of the RSL nucleus which defines each element of the language and an illustration of the symbology. Appendix F represents the TLS kernel which contains the flow and data hierarchies developed as a result of the methodology defined in Section 3.2. Appendix G presents the set of R-NETs and SUBNETs produced by the Calcomp capability of the interactive graphics segment of REVS. Appendix H represents the complete TLS Data Base maintained in the ASSM as extracted by the RADX segment of REVS.

The RSL requirements statements in the Appendices have been produced by REVS from the ASSM in much the same manner that the information content of a software specification would be developed. Editing of this information into a specification document would be adapted to the particular needs of a specific program. A sample PPR specification was produced in Reference [2] and the review it elicited has underscored the need for adaptation of the extracted information to the specifics of an application. Therefore, neither REVS nor SREM is designed to produce a set specification format; this final step is left to the discretion of the user based on the needs of each particular program.

3.0 FUNCTIONAL REQUIREMENTS

Software requirements can be viewed as defining either what must be accomplished or how well an activity must be done. The former is termed a "functional requirement", since it specifies data processing functions; the latter is termed a "performance requirement" since it constrains the quality of performance of the function in the system. In another sense, a functional requirement can be thought of as defining the required output in terms of the available inputs. In a simple case, a program might be named SUMMER and have a functional requirement of generating the sum of a sequence of input numbers (X_i). Defining the output after the ith input to be Y_i, the performance requirement might be that $(Y_{i+1} - Y_i)$ be within ε of X_{i+1}.

Note that while the functional requirement specifies what must be done, and the performance requirement constrains how well it must be accomplished, the means of accomplishment is left to the software process designer. Since the means of implementation is not specified, the requirements are then said to be design-free.

Forms and techniques for representing functional requirements have evolved in recent years and have culminated in the concept of Requirements Networks (R_NETs). Originally, verbal descriptions of functions were attempted, but the verbiage was found to be cumbersome and ambiguous. Later, through Engagement Logic and Functional Flow Block Diagrams (FFBD's), diagrams replaced many words in the requirements specification, unfortunately much of the ambiguity remained. Notably, it was difficult in practice to trace required processing paths; data definitions were incomplete; and these mechanisms did not lend themselves to consistency or completeness analysis.

To resolve the problem of recognizing processing paths, a thread description was attempted; however, the number of threads in a real system proved so large that the (essentially one-dimensional) representation was almost as hard to use as conventional English text. Conversion to the concept of thread trees somewhat reduced the magnitude of the thread problem, but did not of itself resolve the other difficulties of undesired specificity (in AND branches), ambiguity (especially in data), and awkwardness for analysis.

SREM embodies the thread tree (R_NET) concept for specification of processing paths and augments the overall capability in terms of precision and explicitness. Specifically, the desirable properties preserved in the SREM concept of R_NETs were:

- Graphic representation of functional requirements
- Path orientation for specification of threads
- Design (implementation) independence.

In addition, the use of R_NETs permitted inclusion of the following additional desired properties:

- Unambiguous statement
- Analyzable models
- Explicit data specification.

In effect, these six properties became the top-level specification for the tools (REVS/RSL) and methodology (SREM) for functional requirements specifications.

It is significant that the properties carried over from previous approaches are those relating to subjective measures of legibility, utility and design freedom. The added properties are objectively assessible - most readily by demonstration. Thus, a part of the SREM has been the demonstration of completeness, freedom from ambiguity, and other attributes through static analyzers of the explicit (machine-readable) Requirements Statement Language. By expressing functional requirements in machine-readable form, and by using the tools developed on a variety of programs in both industry and academia, it is possible to generate an ultimate test of a functional specification, i.e., a functional simulation. A simulator built without human intervention from specification statements is a total demonstration of the consistency, precision, and completeness of that specification. With a suitable driver, such a simulation provides a mechanism for defining frequency of transactions, and examination of gross aspects of system tradeoffs.

Fundamentally, there are three different ways of conceiving software requirements. The classical approach is functional, i.e., what operations are to be performed by system logic embodied in the software.

Secondly, a thread approach is more nearly mechanical, i.e., what are the interfaces and the properties of the messages communicated through them. The third concept may be termed philosophical; i.e., what are the realities of the world the DPS perceives and what information about those realities must be manipulated. Each approach can lead to mechanisms by which requirements may be generated; SREM uses all three concepts concurrently.

The functional approach is embodied in the concept of a Requirements Network (R_NET), which defines the processing flow required of the software. The mechanical concepts are reflected in the heavy dependence of SREM on definitions of messages through interfaces in establishing the top level of data hierarchies. The philosophical concept is preserved in the implementation-independent hierarchies defined within the concept of entities.

Activities defined by SREM that lead to specification and validation of the functional requirements have been segmented into five phases:

- Phase 1 - Definition of Subsystem Elements
- Phase 2 - Kernel Evaluation
- Phase 3 - Completion of Functional Definition
- Phase 4 - Completion of Management & Control Information
- Phase 5 - Dynamic Functional Validation

The activities and steps within each of these phases are described in the following sections in the context of the Track Loop example.

3.1 PHASE 1 - DEFINITION OF SUBSYSTEM ELEMENTS

There are two different "structural" elements to be defined in the first stage of functional specification. One is the flow connectivity previously represented with Engagement Logic or FFBD's. The other, contained in the current methodology (SREM), defines the required data hierarchies. Previously, a specific methodology was not available for definition of flow connectivity. By adding the data hierarchy to the structures, a step-by-step approach to top-level requirements development has been identified in which only those areas requiring engineering creativity are left unconstrained (however, these areas are clearly identified).

The first-time SREM user may find it necessary to reorient his thought processes to effectively use the methodology, the language and the software tools. In time, however, the phases and steps will form a natural progression for the job to be accomplished. The requirements engineer and management should always keep in mind that their task is to develop requirements for software and not to design the software itself. The pertinent question is to constantly ask, "Is this a precise statement of what the DPS is required to do in the most general way such that the process designer has a maximum range of choices in deriving a solution?"

A typical specification process usually starts with a formulation of the processing steps involved and later considers the data needed to support that processing. SREM partially reverses this order of consideration; the order of concern is 1) "What data are presented to the DPS for processing?", and 2) "What data are expected from the DPS as output?" From the answers to these questions, an initial concept of the required processing steps can be derived.

As a general guideline, it is suggested that the work at each step be completed before proceeding to the next step. In large projects, involving many people, the needs of communication and coordination make this mandatory. In smaller projects, especially those involving one or two people, there is often a rush to do all the steps for one small area of the system before considering the next area of concern. This may be possible for a DPS

problem where the processing functions are independent, but, at best, many valuable insights into the required operation of the DPS as a whole may be lost. At worst, a significant amount of rework may be involved.

3.1.1 Initial Inputs

The initial inputs required for application of SREM are, typically, a system specification and its companion interface specifications. The objective is to generate a complete specification for the data processing subsystem (DPS) from these basic inputs. Usually, at this early stage of system development the input specifications are incomplete, contain many ambiguities, and leave several issues for future resolution. SREM does not require that all omissions be resolved before proceeding. Instead, a user can proceed from that which is clearly defined, and use the facilities of the RSL management segment to spotlight issues needing resolution. Assumptions and decisions may be necessary, often based on inadequate data. These should not be avoided but rather entered in the ASSM using the RSL element DECISION and its attributes. The important thing is to make these entries as they arise. This not only leaves a traceable record of current status for others, but leaves a valuable record of the evolution of critical thoughts about the subsystem for future reference.

The methodology documented here refers to one of many practical starting points for developing software requirements; in particular, it refers to one which is consistent with top-down system evolution. It assumes that a system specification and the key interface specifications have been developed to a significant level-of-detail to enable requirements definition and that the allocation of functions to the subsystems (notably, to the DPS) has been completed. Basically, the methodology is essentially unchanged if the tools are applied either earlier or later in the development process, but the amount of effort needed in each phase, particularly the extent of creativity which must be applied, varies greatly. Consequently, the present methodology is sufficient for the defined starting point and is readily adaptable if the tools are applied either during system definition or after code design (as a V&V tool).

```
SUBSYSTEM:  SSC2
(*I*),
     CONNECTED TO:
          INPUT_INTERFACE:  CC_IN
          OUTPUT_INTERFACE:  CC_OUT.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS.



SUBSYSTEM:  SSPERMFL
(*K*),
     CONNECTED TO:
          OUTPUT_INTERFACE:  DATA_RECORD.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.



SUBSYSTEM:  SSRADAR
(*J*),
     CONNECTED TO:
          INPUT_INTERFACE:  RADAR_CLOCK_IN
          INPUT_INTERFACE:  RADAR_IN
          OUTPUT_INTERFACE:  RADAR_OUT.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_1_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_1_PERFORMANCE_REQUIREMENTS.
```

Figure 3-1 RSL Subsystem Entries

3.1.2 Interface Definition

The first RSL entries in the ASSM are concerned with identification of the DPS interfaces which CONNECT with other subsystems. Usually, these are the elements which are most clearly defined in the originating specifications. Also, it has been found that the interfaces, and the messages passing through them, are the key focal points for progressive development of the requirements structure.

Consider the TLS specification (DPSPR) and interface specifications (IFSs) contained in Appendices A, B, and C. These documents refer to two subsystems which interface with the DPS, namely the Radar and C^2 (Command and Communications). C^2 will be referred to as a subsystem for convenience, even though it is a separate system, external to the TLS. A closer examination of the DPSPR reveals that the DPS is to output data to permanent files. Although not explicitly required by the specification it will later become apparent that it is conceptually useful to define permanent storage as a third subsystem separate from the DPS, even though it is embedded in the DPS. For REVS use, these three subsystems have been designated as SSRADAR, SSC2, and SSPERMFL, respectively.

In the specifications, three separate interfaces between the DPS and the Radar are mentioned. Through one input interface the radar sends returns to the DPS; through another it sends clock inputs to the DPS. The DPS issues commands to the radar through an output interface. These interfaces are designated as RADAR_IN, RADAR_CLOCK_IN, and RADAR_OUT, respectively. The names are arbitrary, but should be meaningfully related to specification terminology. Note that an interface is designated as "input" or "output" from the viewpoint of the DPS.

Similarly, the specifications identify one input interface between C^2 and DPS. (Denoted as input interface CC_IN). Data recorded by the DPS in permanent storage apparently are never accessed from that source during DPS operation. Therefore, the DPS is linked to the conceptual subsystem SSPERMFL via an output interface, DATA_RECORD.

At this point, the subsystem and interface definitions, and their relationships, can be coded in RSL for entry into the ASSM. Figure 3-1 illustrates one way of expressing these data in RSL. Note that the

management segment attribute, DESCRIPTION, has been used to explain the nature of SSPERMFL. This entry is for example purposes here, and will not be perpetuated in the TSL example. However, notations such as this have value in a real system development project and should be encouraged.

Also note that the DPS itself is not entered into the ASSM as a SUBSYSTEM, since it is inherently the object of all software requirements.

3.1.3 Message Definition

Having defined the subsystems, the interfaces, and their connections, the next step is to examine the discrete blocks of data, or MESSAGEs which are PASSED BY the interfaces. A MESSAGE is MADE BY its contents, whether they be DATA or FILEs.

Using SREM, the concepts of "message name" or "message category" differ from that of "message type". In RSL, the identifier associated with MESSAGE refers to the message name or category. MESSAGEs are distinguished by differences in their data contents. Thus, two blocks of data with different data contents, which pass through an interface, must be defined as different MESSAGEs, hence must have different message names. The message name is not contained in the data, it is an external label.

On the other hand, two packets of data may have identical data content with different values, and may require different processing to be done on the data within the DPS. In this case there would be two instances of the same MESSAGE, but with different "message type". Further, it is typical for one of the data elements in the message to be a message type identifier, with a unique value for each type. Otherwise, the DPS could not distinguish between types of messages and perform different processing operations on each type. Messages with different names must also have a unique identifier in order for the DPS to distinguish between messages. It is most economical to require that all messages, whether of different name or type, contain a type identifier as a data element with a unique value, different from the message name. This avoids both mental confusion and possible misinterpretation by the RSL translator.

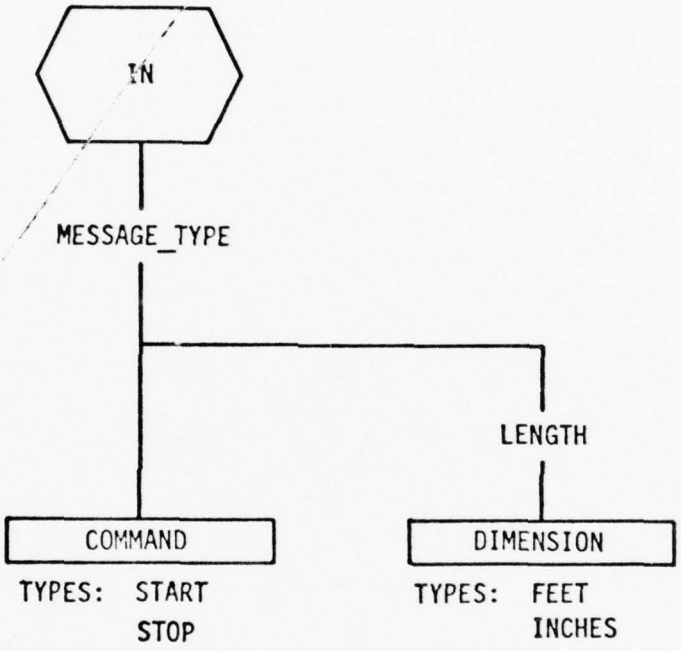


Figure 3-2 An Elementary Interface Hierarchy

To illustrate these concepts, consider a DPS with one input interface called IN. Across this interface pass four distinct types of messages. The first two types consist of a single data element with different values: START, STOP. These command the DPS to start and stop processing, respectively. The second two types consist of a data element with two values: FEET, INCHES. This element defines which units the real number is in. Further, the DPS is to convert the input data to meters for further processing.

These messages can be distinguished by defining a data element MESSAGE_TYPE with four enumerated values: START, STOP, FEET, INCHES. For the types FEET and INCHES, an associated data element, LENGTH, can be defined. Over the four message types two categories can be identified according to data content and function. The first category consists of messages with only one data element, MESSAGE_TYPE, which provides commands to the DPS. The second category consists of messages with two data elements, MESSAGE_TYPE and LENGTH, which provide dimensional input data to the DPS. The first category is named a COMMAND message; the second category is a DIMENSION message. Hence, a COMMAND message has two types: START and STOP. A DIMENSION message also has two types: FEET and INCHES. This latter message could not be called LENGTH because that name is assigned to one of the data elements within the message.

The message names, types, and data contents associated with the various interfaces must be extracted from the system and interface specifications, often from widely scattered locations. Sometimes the specifications only define message types, and the requirements engineer must group these into categories which will be named MESSAGEs in RSL. For these reasons, some pencil-and-paper preliminary work is usually needed to organize the data before translating them into RSL inputs. One diagrammatic representation of the messages passing an interface, that has proved useful, is shown in Figure 3-2 for the simple example discussed above.

The diagram is simply a tree originating at the interface. Each branch of the tree terminates in a box corresponding to a MESSAGE name. Data elements common to all messages are placed on the tree before the first branch point. Data elements unique to a specific message are placed on the branch unique to that message. Elements common to two or more

messages, but excluded from others, are placed on an intermediate branch leading only to the appropriate messages. (The TLS example discussed below will illustrate this latter condition.) The data element names may refer to data items, or files. It is useful to note files as such on the diagram. A file is a collection of instances of data items, each instance having the same structure. If desired, the types of each message can be noted below the message name. While the diagrammatic format shown has proven useful, it is in no way mandatory. Whatever notation or format aids a specific problem should be used. The important point is to organize the material on paper before writing the RSL inputs.

A useful rule-of-thumb for using SREM is: don't define details until they are needed for the purposes of the moment. At this stage of an analysis, interest is focused solely in the elements which are common, and unique, to the various message types. No further detail is needed. For instance, if it is known that an identifiable group of data is unique to a given message, it is only necessary to name the group as a data item on the tree. In practice this is often easy, because the exact composition of the group may be ambiguous long after the group itself is identified. In any case, the composition of a group is easily defined by the RSL relation INCLUDES when that level of detail is needed.

Now, consider the messages in the TLS example, starting with those coming from the C^2 "subsystem". Paragraph 3.2 of the TLS C^2/DPS IFS states that four types of messages are transmitted from the C^2 to the DPS:

- Initiate Engagement Mode
- Terminate Engagement Mode
- Handover Image
- Drop Image Track

Implicitly common to all these message types is some sort of message type identifier. Since these are all common messages they were assigned a single identifier (COMMAND_ID). Paragraphs 3.2.1 and 3.2.2 of the IFS imply that there are no other data elements in the first two message types. Both of these types have a common function, namely to command a change in the operating mode of the DPS. Thus, they form a single MESSAGE category

(named MODE_CHANGE). Hence, as shown in the diagram of Figure 3-3, there is a message MODE_CHANGE, of two types, with a single data element, COMMAND_ID, which distinguishes the two types.

The remaining two message types contain other data elements, in addition to COMMAND_ID, as stated in paragraphs 3.2.3 and 3.2.4. Common to both is an element called variously "image designation", or "image designator", but which is obviously a single element called HO_ID (shortened from HANDOVER_ID). This additional element completes the definition of the "Drop Image Track" type message. Since the data content of this message is unique, it forms a message category by itself. The identifier TERMINATION was chosen in order to reserve DROP_TRACK for use as an enumerated value of COMMAND_ID.

The remaining message type is also in a category by itself, in this example named a HANDOVER message. In addition to COMMAND_ID and HO_ID, paragraph 3.2.3 of the IFS states that this message contains a data element "image estimated state". However, paragraph 1.1.2.1a of the DPSPR refers to an "estimate of state", while paragraph 1.2.2.1g states that "each handoff shall consist of a unique designator, the state vector, and its covariance matrix." At this point the data element could be defined as INITIAL_STATE_ESTIMATE. But, it seems worthwhile to state the main components, since they are not stated in the IFS paragraph where one would expect to find them. Thus, two data elements are defined, INITIAL_STATE and INITIAL_COVARIANCE, to represent the vector and matrix, respectively. The prefix "initial" is used to avoid confusion with state data generated by the DPS in the course of subsequent processing. Note that there is no need, at this point, to define the vector components and matrix elements individually. The definition of messages related to the CC_IN interface, as shown in Figure 3-3 have now been completed. These messages can be defined in RSL as shown in Figure 3-4. Although not shown here, it is useful to use the capabilities of the management segment of RSL to note the source of the state data definitions in the handover message, and to point out that the IFS is incomplete or at variance with the DPSPR.

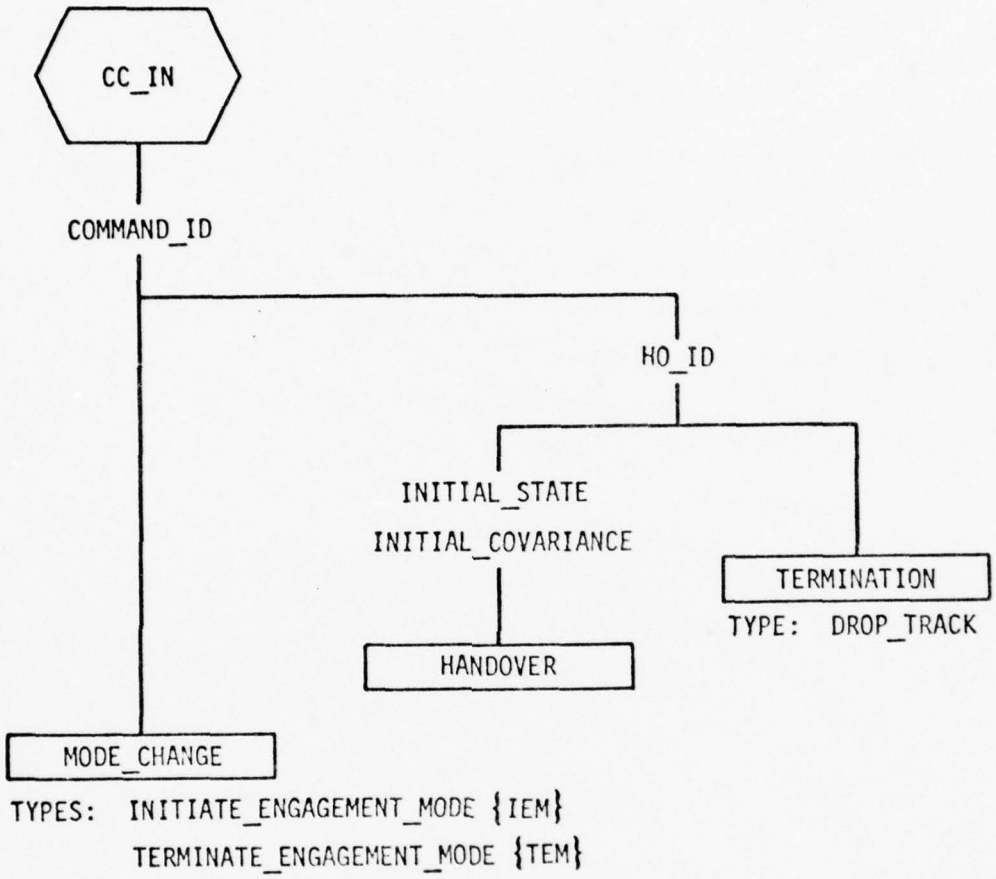


Figure 3-3 C^2 Input Hierarchy

```
INPUT_INTERFACE:  CC_IN.
     PASSES:
          MESSAGE:  HANDOVER
          MESSAGE:  MODE_CHANGE
          MESSAGE:  TERMINATION.


MESSAGE:  MODE_CHANGE.
     MADE BY:
          DATA:  COMMAND_ID.


MESSAGE:  HANDOVER.
     MADE BY:
          DATA:  COMMAND_ID
          DATA:  HO_ID
          DATA:  INITIAL_COVARIANCE
          DATA:  INITIAL_STATE.


MESSAGE:  TERMINATION.
     MADE BY:
          DATA:  COMMAND_ID
          DATA:  HO_ID.
```

Figure 3-4 RSL Message Entries

3.1.4 The Interface Data Hierarchy

At this point the elements of one of the two major data hierarchy types associated with SREM have been partially defined. This is the "interface data hierarchy" depicted in Figure 3-5.

A SUBSYSTEM is CONNECTED TO either an INPUT_INTERFACE or an OUTPUT_INTERFACE which PASSES blocks of data called MESSAGEs. A MESSAGE is MADE BY individual items of DATA, and/or by a group of DATA which INCLUDES individual DATA items, and/or by FILEs. In turn, a FILE CONTAINS individual DATA items. The RSL concept of FILE is more general than the usual software connotation. It is a collection of instances of data, each instance having the same content of data items, without regard to the details of storage, and without ordering unless specified.

In general, a data item must have a different DATA name in each hierarchy in which it appears, even though the different names refer to the same information. The exception is that DATA or FILEs may exist in more than one MESSAGE due to the fact that only one MESSAGE can be active in the REVS system at any particular time. Therefore, the assembly of DATA and FILEs into a MESSAGE is unambiguous regardless of the number of MESSAGEs that may be MADE BY that element. For example, a MESSAGE PASSED THROUGH an INPUT_INTERFACE only exists at the instant of passage (i.e., the enablement of the interface network). An ALPHA accesses not the MESSAGE but the DATA it contains; there can be no ambiguity among those DATA items regardless of the number of MESSAGEs which might contain them, since there can be no more than one MESSAGE entering the system for a given enablement.

SREM is heavily oriented toward an orderly analysis of the interface data hierarchies, in a "top-down" direction, as the first step in defining DPS requirements. This is a natural direction, as the interfaces and the messages crossing them are usually the most clearly defined elements of the originating specifications. As the data definitions are developed in progressively greater detail, the definition of specific processing steps, or ALPHAs, begins to emerge, as well as the processing flow STRUCTURE which links the ALPHAs.

For INPUT_INTERFACES, the "top down" consideration of the data hierarchy follows the flow of processing. For OUTPUT_INTERFACEs, the "top down"

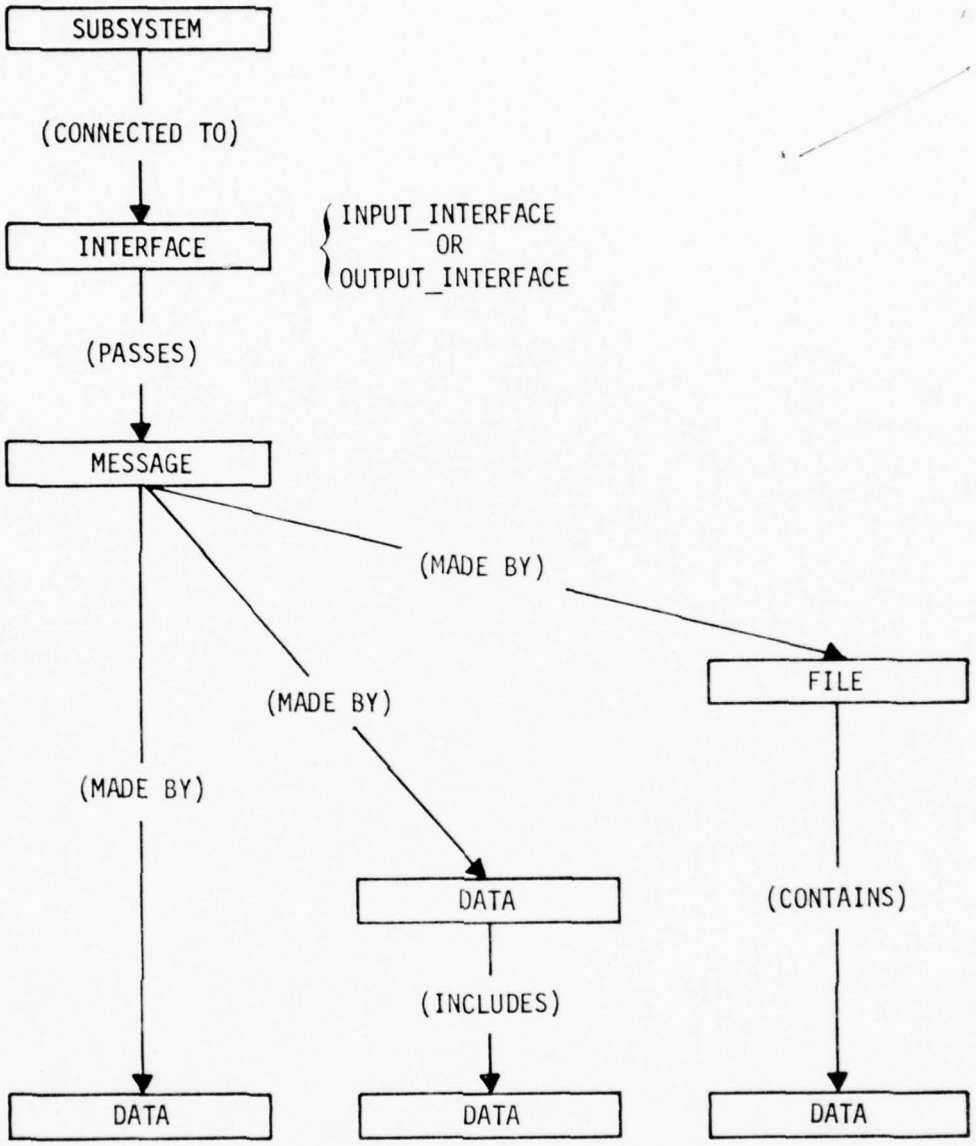


Figure 3-5 Interface Data Hierarchy

direction is opposite to the processing flow. However, backward tracing from the output interface is a valuable tool in constructing the steps necessary to form an output MESSAGE.

Initially, the internal processing required of the DPS may be ill-defined and ambiguous. The requirements engineer will need to draw heavily on experience to synthesize the connections between input and output. As a broad guideline, the primary concern is, "What must be done to the data that is input in order to provide the required output?"

3.1.5 Problems of Definition

In general, specification documents written in English text contain statements which yield different meanings depending on a particular interpretation. Confusion and ambiguity usually result when the requirements engineer encounters specification statements similar to subparagraphs 3.2.5 and 3.2.6 of the C^2/DPS IFS. These two innocent subparagraph titles lead to a number of confusing questions, which are typical of preliminary specifications.

The titles of subparagraphs 3.2.1 through 3.2.4 correspond to clearly defined C^2 to DPS message types, and the content of the text addresses those types. The contents of 3.2.5, titled "Message Acknowledgement", and 3.2.6, titled "Error Handling", are TBS (To Be Specified). Do these titles indicate additional input message types, in conflict with the clear definition in paragraph 3.2? If so, what message is being acknowledged by "Message Acknowledgement"? No messages from DPS to C^2 have been defined, and no requirement for a DPS output interface to the C^2 system is indicated. In fact, Figure A-1 of the DPSPR (Appendix A) clearly indicates that message traffic is one-way, from C^2 to DPS.

On the other hand, a requirement for the DPS to acknowledge receipt of any of the four defined C^2 to DPS message types by transmitting a reply to C^2 may be intended. If so, both the DPSPR and IFS must be modified to reflect this, without ambiguity. Similarly, "Error Handling" might refer to either a message type, or to processing in response to erroneous messages. Resolution of these questions is important because major differences in DPS definition result from the alternatives.

BEST AVAILABLE COPY

Work can either be halted until the problems are resolved, or can proceed tentatively with the assumptions which make the most sense. If the work continues, the problem should be noted in the ASSM to identify what elements are affected by the assumptions. Figure 3-6 shows one way of doing this, for the message acknowledgement problem, using features of the RSL management segment. These entries announce the problem and indicate changes needed later if the assumptions are wrong.

```
DECISION: MEANING_OF_MESSAGE_ACKNOWLEDGEMENT.
  PROBLEM: "IMPLICATION OF CC TO OPS IFS PARAGRAPH 3-2-5
            IS NOT CLEAR. REVISION OF THIS PARAGRAPH AND/OR
            OPSPR FIGURE 1-1 IS NEEDED.:.
  ALTERNATIVES:
    "1. INTERPRET MESSAGE ACKNOWLEDGEMENT AS A MESSAGE TYPE
        FROM CC TO OPS.
     2. INTERPRET AS A MESSAGE FROM OPS TO CC IN RESPONSE
        TO EACH CC TO OPS MESSAGE. THIS NEEDS A OPS
        OUTPUT INTERFACE TO CC.".
  CHOICE:
    "PROCEED WITH ALTERNATIVE 2 PENDING FORMAL RESOLUTION.:.
```

Figure 3-6 RSL Decision Entry

A possible objection is that making these entries and recording these questions is tedious and takes up time. True. However, more time will eventually be taken up by other people asking the same questions (over and over again). Worse yet, others may make different assumptions or fail to detect the ambiguities, resulting in more time spent later in rework. With the information recorded in the ASSM, the answers are available to everyone - even to those who haven't yet thought of the questions.

It is not within the scope of this manual to explore all the aspects of traceability and uses of the RSL management segment. For further development of the TLS example, assume that the problems have been resolved as follows:

- A message called ACKNOWLEDGEMENT consisting of one data element, COMMAND_ID, is required.

- This message is passed from DPS to C^2 via a new output interface, CC_OUT.

- "Error Handling" refers to error processing required of the DPS when a message from C^2 is not identified as one of the four specified types.

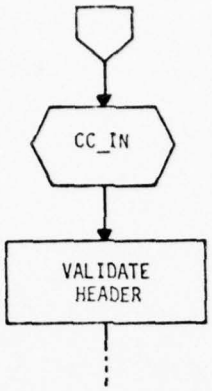
3.1.6 R_NET Definition

A Requirements Net, or R_NET is used to describe the required flow of processing in response to a single stimulus which ENABLES the net. This stimulus may be either the passage of a MESSAGE through an INPUT_INTERFACE, or an EVENT defined by its arrival at a node on the subject R_NET or on some other R_NET associated with the DPS.

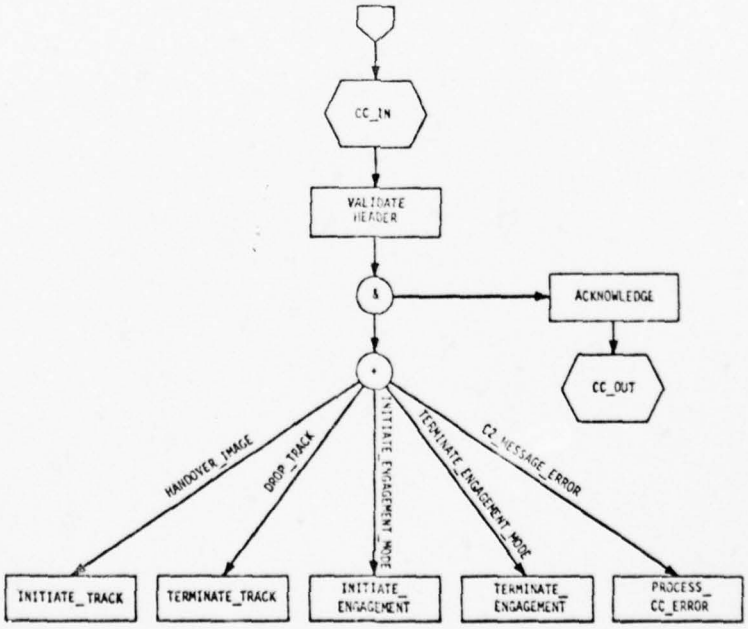
Each INPUT_INTERFACE must enable an R_NET. Otherwise, DATA in a MESSAGE passing the interface could not be processed by the DPS. Hence, there must be at least one R_NET for each INPUT_INTERFACE, since only the processing on an R_NET can distinguish between the arriving messages. Thus, the first logical step in defining the R_NETs of the DPS is to define one for each INPUT_INTERFACE. In the TLS example, three such interfaces have been previously defined, so it is necessary to develop three corresponding R_NETs.

R_NETs may be entered into the ASSM manually from the Anagraph terminal, or via a card deck of RSL statements. In most cases, the net should be diagrammed on paper first to develop the concept fully and minimize revisions.

As a first example, consider the R_NET ENABLED by INPUT_INTERFACE CC_IN. The Input-Interface is the initial node on the net. It is reasonable then, but not mandatory, to provide an ALPHA for common processing of all MESSAGEs through that interface, for such purposes as validating data common to all MESSAGEs. Thus, a typical starting structure is:



It was noted during MESSAGE definition that each was distinguished by the differences in their data contents. Since the input to an ALPHA is fixed, there must be a unique ALPHA for each MESSAGE. There also may be several message types for each MESSAGE, and it is reasonable to expect different processing, hence different ALPHAs, for each type. While not always true, this is usually a profitable assumption at this stage of R_NET development. Further, an additional ALPHA is usually needed for error processing of messages not recognized as one of the valid types. Therefore, it is possible to draw the following skeleton associated with an input interface with the information gained from our preceding analyses of the specifications. (Refer to Appendix D, Figure D-1 for symbols and allowable structures.)



The OR node with multiple branches, as shown, is described in RSL with the aid of the CONSIDER phase. The object of consideration is a data element with an enumerated set of values, in this case the data element COMMAND_ID. If the value of COMMAND_ID in a particular message does not match one of the four valid MESSAGE types, then the C^2_MESSAGE_ERROR branch is invoked.

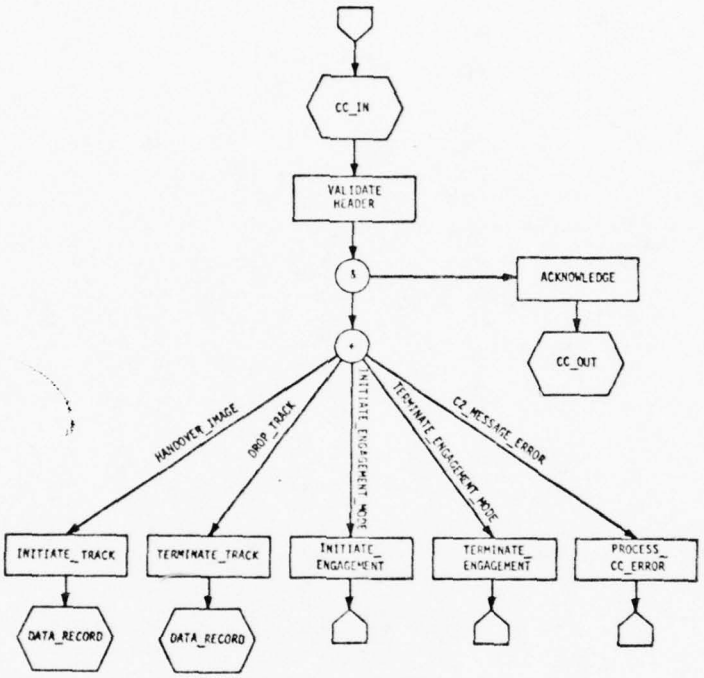
The ALPHAs defined above reflect all processing required for a given message type. The names chosen reflect a gross conception of the nature of the processing. They are subject to change. As analysis proceeds the definition of the R_NET will be refined. The first tentative ALPHAs may expand, and the branching structure will be modified for all but the simplest systems. Hence, there should be no rush to enter an R_NET into the ASSM at the first opportunity. This activity is best performed when the definition of the net has stabilized and possible relations with other R_NETs are identified.

Where the previous effort was purely mechanical, it is now necessary to apply some creativity to complete the interface network. There are two fundamental approaches to that completion from the available documentation, i.e., thread tracing and sentential analysis. Both should be used so that completeness of statement is assured.

The first operation is thread tracing. By reading the source specifications, the processing steps required may be traced from an input port to their logical termination. When all paths have been traced, not only for the input networks but also for those developed in the following paragraphs, the set of processing steps (ALPHAs) required should be complete. Sentential analysis provides a cross-check by separating each specification sentence into its nouns (which correspond to system data) and its verbs (which correspond to ALPHAs). The two sets of ALPHAs should be identical; if not, they are made to be through refinement of the diagrams.

The result of specification analysis is completion of the paths from each INPUT_INTERFACE. For example, there is a requirement in the TLS that each MESSAGE received from C^2 be acknowledged. Therefore the AND node is added, an ALPHA is provided to FORM the MESSAGE: ACKNOWLEDGEMENT, and the appropriate OUTPUT_INTERFACE: CC_OUT is indicated. Continuing this

process developed the following diagram. It is a complete R_NET for CC_RESPONSE requirements for TLS except that it does not yet reflect inter-network connectivity. Note that each branch ends at either an OUTPUT_ INTERFACE, or at a TERMINATE symbol.



In tracing input networks, many of the MESSAGEs to be output by the software will have been isolated. However, not all messages for any OUTPUT_ INTERFACE, nor indeed any MESSAGE for some of them, may be defined. Thus, there is an inverse operation for output networks which traces back from an OUTPUT_INTERFACE through individual ALPHAs for each possible output MESSAGE to the earliest operation required for it in the specifications.

This procedure is not necessary for the network above. All MESSAGEs passed by CC_IN require an ACKNOWLEDGEMENT message response to be passed by CC_OUT. All HANDOVER messages clearly require a TRACK_INITIATION message to be passed by DATA_RECORD. The above network completely describes the only conditions where these responses are generated within the DPS. A TRACK_TERMINATION message is passed by DATA_RECORD in response to an input TERMINATION message passed by CC_IN. However, this case represents only

one condition where a TRACK_TERMINATION message is generated by the DPS. The remaining conditions will occur on other R_NETs. But, all of these responses have one thing in common. Each is a single MESSAGE passed through a specific OUTPUT_INTERFACE in response to a given MESSAGE or class of MESSAGEs passed by a single specific INPUT_INTERFACE. These are examples of "synchronous processing": the input is joined to the output by a direct path of processing steps, and none of the data used are modified by processing performed on any other path.

In the context of R_NETs, logical connectivity is maintained, not only by a continuous path through one R_NET, but perhaps through additional R_NETs by means of EVENTs. An EVENT is an alternate means for enabling an R_NET. A single logical path is formed by the path leading to the event on the enabling R_NET and continuing on a path on the enabled R_NET. Such paths represent "synchronous processing" only if none of the data used are modified by an independent path.

"Asynchronous processing" is a more complex concept to grasp. This type of processing is implicit when two R_NETs are related by data, but without the "logical connectivity" represented by flowing tokens. An example involving three simple R_NETs will serve to illustrate the basic forms of "asynchronous processing." The example is defined in Figure 3-7.

Whenever a subsystem SS1 passes an XIN message through the DPS interface SS1_IN, the R_NET named X_VALUE is enabled. This R_NET accepts the data NEW_X in the message and, if NEW_X satisfies certain conditions, updates the value of a global variable, X, to the value of NEW_X. Whenever a second subsystem, SS2, passes a YIN message through the interface SS2_IN the R_NET named Z_VALUE is enabled. This R_NET accepts the data Y in the message and adds it to the current value of X defined in the data base to form Z (defined as a global variable for this example). The value of Z is contained in the ZOUT message sent to SS2 via SS2_OUT, but is also retained in the DPS because Z is defined as global data. Further, a third R_NET named ZZ_VALUE periodically enables itself, to compute Z^2, by means of an event on its own structure and an associated time delay. The value of Z which is used is, of course, that which is current in the data base. There is no specific order of enablement required of the R_NETs.

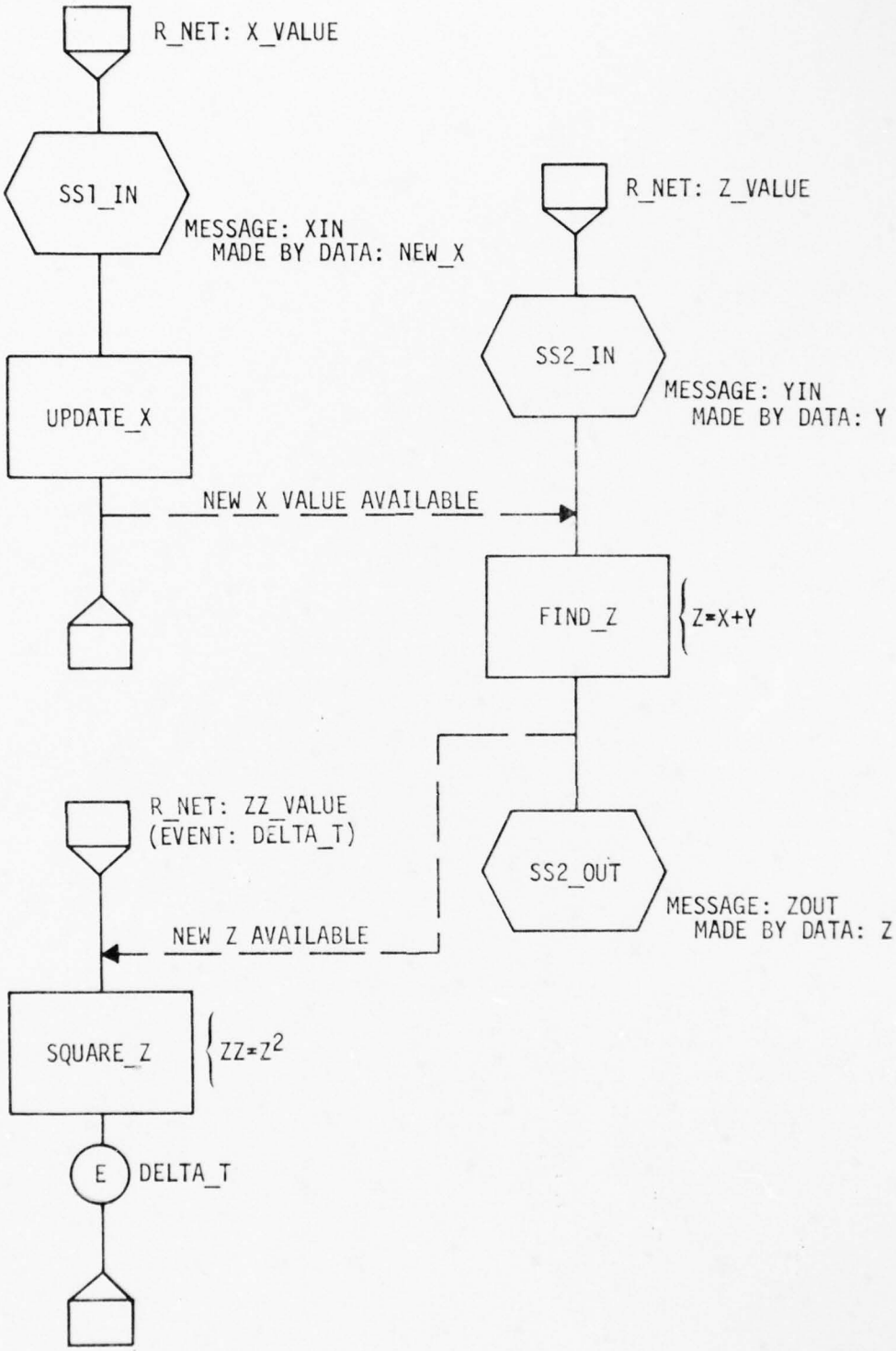


Figure 3-7 Three Asynchronous R_NETs

Examination of this system shows that the values of Z and Z^2 depend not only on the inputs X and Y (or their previous values), but also on the sequence of enablement of the R_NETs. Thus, within a given time interval, there is not a one-to-one correspondence between the latest values of X and Y and the latest values of Z and Z^2. For that matter, no such correspondence exists between Z and Z^2.

A simple example of "asynchronous processing" in the TLS example might be the use of radar clock time. The sole function of the R_NET called RADAR_TIMING is to accept timing inputs from the radar and update the global variable RADAR_CLOCK. If another R_NET needed radar time, it would use the value of RADAR_CLOCK. This value does not reflect the current time, but rather the time at which the radar formed the message which was last accepted by the R_NET called RADAR_TIMING.

The major, and most complicated, example of "asynchronous processing" in the TLS is the relationship between radar returns and the next set of radar commands. Synchronous tracking would require that data from the last radar return from an object be used to produce the next succeeding radar order related to that object. Asynchronous tracking allows use of the last data available in the DPS, even though more current data may be coming from the radar. Asynchronous tracking allows less stringent DP response times, better time-line usage, and permits gradually degraded response with increased system load. Synchronous tracking, on the other hand, places stringent constraints on the DP which lead to saturation and loss of track under relatively light loads.

The definition of an asynchronous processing concept is subtle and often difficult. No cookbook solutions can be offered for this creative process. First, the R_NETs must be traced both forward from the input interfaces and backward from the output interfaces. The data and logical connectivity to fill the gaps between must then be added through an active and creative engineering thought process. Note that SREM does provide a framework of organization which fosters consistency, and ultimately leads to a simulation which can detect errors of concept.

3.1.7 Entity Definition

One of the most powerful concepts used in SREM is that of an "entity". This concept allows a clear expression of the role of the DPS requirements in the same operation, and the RSL facilities provided tend to enforce that perspective. An "entity" is simply a thing, or category of things, in the external world about which the DPS must collect, process, and maintain data. Entities are closely tied to the reasons for the existence of the system and its DPS. They are usually implicit in the wording of the originating specifications.

For instance, the basic purpose of the TLS is to gather data on the position and velocity of designated objects within its detection range, and to predict the position and velocity of those objects at some future time. This suggests that "objects" might be an entity. However, "designated objects" would be a better candidate, because the TLS is not expected to detect any objects other than those the C^2 System orders it to track.

Considering the TLS as a whole, this might be a reasonable choice. But the focus is on the DPS and it is not "aware" of objects because it does not perceive them directly. The radar performs the sensor functions. Thus, the DPS is only aware of what the radar perceives as objects and reports to the DPS. The nomenclature of the DPSPR calls these "images". Further reading of the DPSPR shows that much of the DPS processing is concerned with the proper classification of these images, and eventual elimination of those which do not correspond to real objects (ghosts), or which are redundant images of the same object. During the time that an image is considered an "image in track", a certain instance of data items must be maintained in the DPS and be associated with the proper image. When the DPS decides that the image is probably redundant or a ghost, or should drop track on that image for other reasons, the DPS must maintain, for a time, a different set of data about that image.

Thus, there is a general category of things called "images" which are of importance to the DPS. In SREM, such a category is called an ENTITY_CLASS. Hence, for TLS an ENTITY_CLASS named IMAGE was defined. There are two types of IMAGE, distinguished by the different data associated with each type; designated as IMAGE_IN_TRACK, and DROPPED_IMAGE. Each IMAGE of

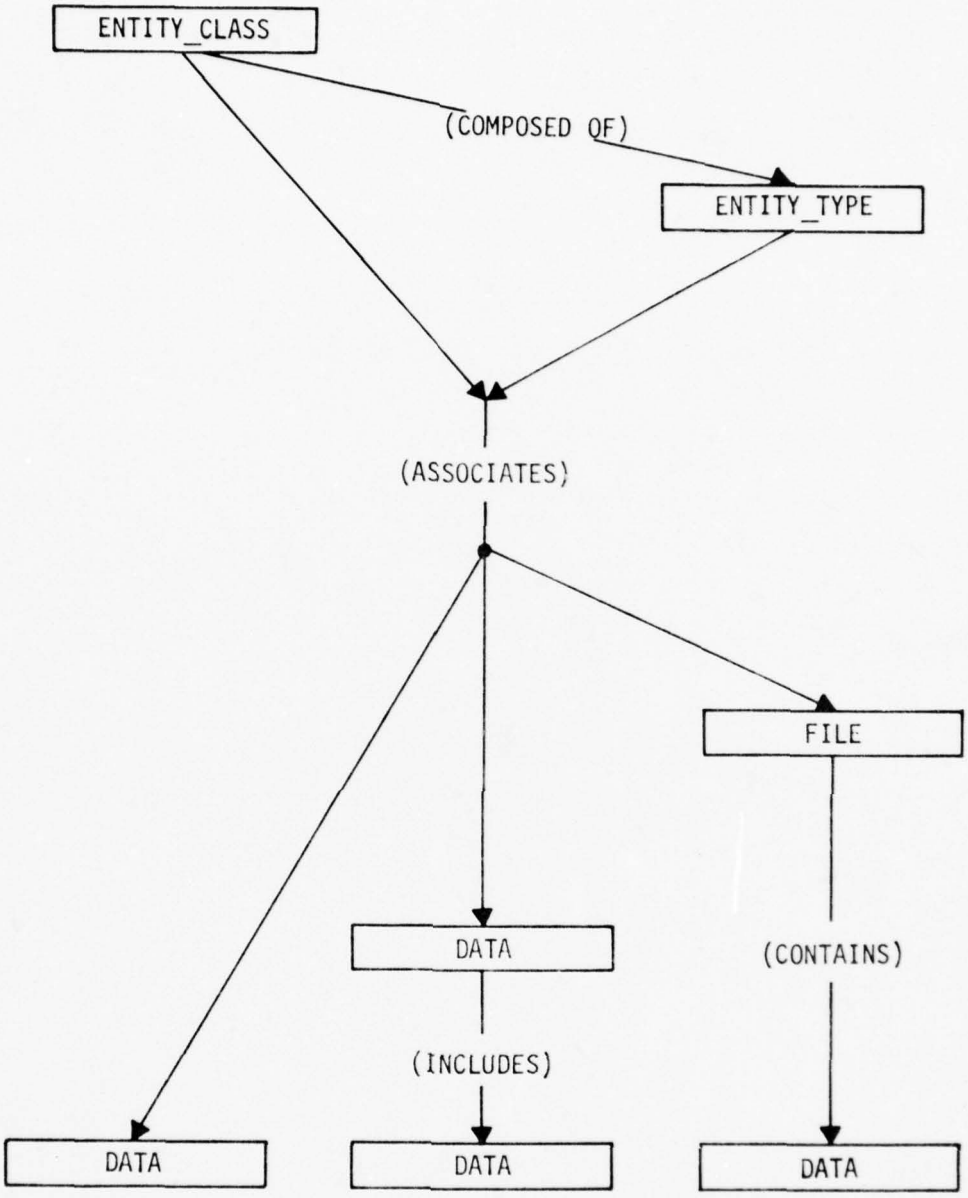


Figure 3-8 Entity Data Hierarchy

which the DPS is aware has an instance of data uniquely associated with it. This instance may be composed of DATA items and FILEs. The composition of the instance is a function of ENTITY_TYPE and, by definition, should be different in some manner from at least one other type in the class. However, data common to all types may be associated with the ENTITY_CLASS itself. Multiple ENTITY_TYPEs may have identical data associated with them. These usually imply a transition from a common earlier state (e.g., an ENTITY_TYPE I is set to either ENTITY_TYPE J or ENTITY_TYPE K depending on some decision in the DPS).

A second ENTITY_CLASS is defined in TLS for radar PULSEs. The pulse is an external phenomenon (an electromagnetic signal) about which the data processor is found to need data, and which exists in multiple copies. Therefore, it satisfies the criteria for consideration as an ENTITY_CLASS. The fact that data must be 'remembered' about each pulse while it is in transit, and the required data themselves, derive from the IFS through the process of defining the functional requirement. Thus, when considering the determination of range to the target, it is necessary to know both the start time of the range gate relative to the start of transmission and the time within the gate that the signal was detected. The IFS asserts that the radar return contains the time within the gate; the start time of the gate must therefore be 'remembered' by the data processor from the command which gave rise to the return. Thus, the data required on a pulse in transit have to do with the transmission parameters relevant to different pulse waveforms. Consideration of data and logical usage differences leads to the definition of four ENTITY_TYPES named T1_T2, T3, RETURNED_PULSE, and LOST_PULSE within the ENTITY_CLASS: PULSE.

3.1.8 The Entity Data Hierarchy

The second major data hierarchy associated with SREM is the "entity data hierarchy" depicted in Figure 3-8. The means for manipulating data contained in an entity hierarchy differ from those used with the interface hierarchies described in Section 3.1.4.

An ENTITY_CLASS is COMPOSED OF one or more ENTITY_TYPEs. If there are data elements common to all ENTITY_TYPEs, then the ENTITY_CLASS ASSOCIATES these DATA items and FILEs. For data elements specific to an entity type,

the ENTITY_TYPE ASSOCIATES the applicable data elements. Once again, the DATA item is the lowest element in the hierarchy. Each FILE CONTAINS items of DATA.

The chief characteristic of the entity data hierarchy is the unique method of manipulating data which is implemented in RSL. There is no way, in RSL, for a user to specify that a FILE be created or destroyed by an ALPHA. FILEs may only be modified by ALPHAs (although instances in files are created or destroyed in the BETA models). RSL provides a mechanism to require that an ALPHA CREATES or DESTROYS the knowledge that an instance of an ENTITY_CLASS exists in the environment. When an ALPHA creates a new instance of an ENTITY_CLASS, a new instance of all the common DATA and FILEs ASSOCIATED WITH that class is initialized. The ALPHA may then assign proper values to those data elements through its BETA definitions. These data elements are retained throughout the life of the instance.

The ALPHA can also SET the ENTITY_TYPE. When this is done, an instance of all the specific DATA and FILEs ASSOCIATED WITH the ENTITY_TYPE is initialized and can be assigned proper values within the BETA definitions. When the instance of ENTITY_CLASS is SET to a new ENTITY_TYPE, the specific data elements unique to the old type are destroyed and a new instance of data elements specific to the new type is initialized.

As the data processing system gathers information about an entity, it may first identify the entity as being of one ENTITY_TYPE, and then another. Instances of a class of entities thus evolve from one type to another, but instances of one class (e.g., IMAGEs) can never evolve into another class (e.g., PULSEs). Each ENTITY_TYPE therefore COMPOSES just one ENTITY_CLASS. An instance belongs to one ENTITY_CLASS after an ALPHA CREATES it, and it belongs to the last ENTITY_TYPE to which an ALPHA SETS it. Eventually, an ALPHA may destroy (knowledge of) the instance, and all data associated with the instance vanish. Although the RSL statements for CREATES and DESTROYS refer to the name of the ENTITY_CLASS, the operation is applicable only to a single instance.

This concentration of the requirements for creation and destruction of knowledge about external entities, rather than the mechanics of data structures, allows the user to focus on the requirements for the DPS as a

part of the system in which it is embedded. Otherwise, the tendency would be to stray into process design issues, such as when to set up FILEs.

As was done with MESSAGEs in Section 3.1.3, it is useful to diagram the entity data hierarchies before coding them in RSL. Figure 3-9 shows diagrams of the TLS hierarchies associated with the two ENTITY_CLASSes, PULSE and IMAGE. When the definitions are stable and ready to enter in the ASSM, the entries can be coded in RSL as shown in Figure 3-10 for the ENTITY_CLASS: IMAGE.

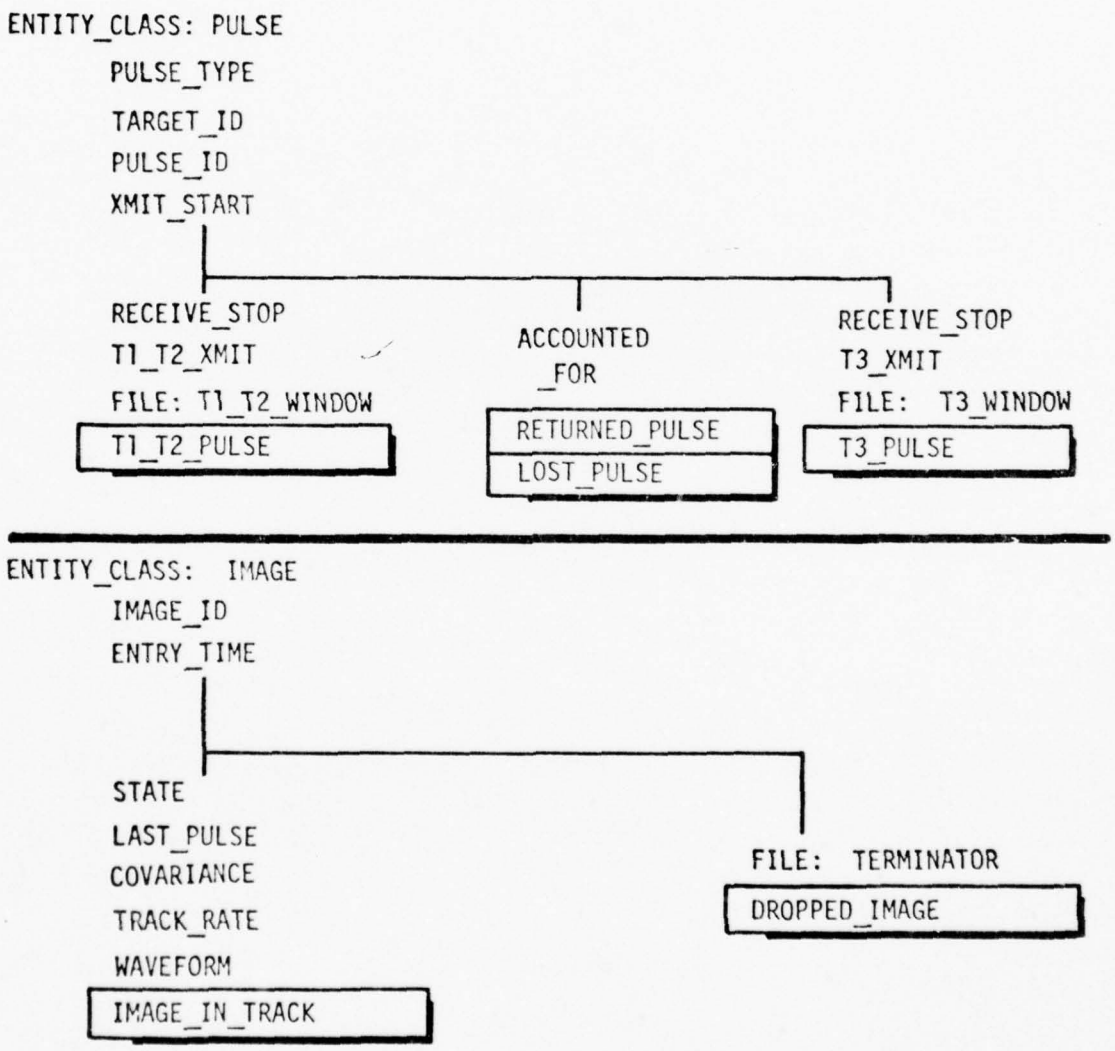


Figure 3-9 Entity Hierarchy

```
ENTITY_CLASS:  IMAGE.
     ASSOCIATES:
          DATA:  ENTRY_TIME
          DATA:  IMAGE_ID.
     COMPOSED OF:
          ENTITY_TYPE:  DROPPED_IMAGE
          ENTITY_TYPE:  IMAGE_IN_TRACK.
ENTITY_TYPE:  IMAGE_IN_TRACK.
     ASSOCIATES:
          DATA:  COVARIANCE
          DATA:  LAST_PULSE
          DATA:  STATE
          DATA:  TRACK_RATE
          DATA:  WAVEFORM.
ENTITY_TYPE:  DROPPED_IMAGE.
     ASSOCIATES:
          FILE:  TERMINATOR.
```

Figure 3-10 RSL Entity Entry

3.1.9 Independent FILEs

When defining processing requirements, there may be a need for data sets which fit in neither a transient interface data hierarchy nor a permanent entity data hierarchy. These should have the properties of multiple instances of a data item or data group and which either need to be 1) retained as GLOBAL data, or 2) ordered in some particular way, or 3) temporarily maintained as a class of data meeting some selection criteria. These needs can be satisfied by defining an independent FILE. This FILE exists in the DPS at all time, even though it may be empty. Although instances in the file can be created and destroyed

within BETA and GAMMA executable descriptions of ALPHAs, the FILE itself cannot be created or destroyed. It can only be modified. FILEs can be INPUT TO ALPHAs or OUTPUT FROM ALPHAs or both.

The distinction between the concepts of ENTITY_CLASS and independent FILE often may appear fuzzy. The key distinction is that a FILE is a set of data, an ENTITY_CLASS is the subject of data.

In the TLS, there exists a file of constants called WAVEFORM_TABLE. This FILE is part of the site-adaptation data for the system, and is independent of real-time input. Two other TLS examples of independent FILEs are those called COMMAND and CANDIDATE. These files are dynamic (i.e., change in response to real-time data). Both of these files are used to select instances from ENTITY_TYPE: IMAGE_IN_TRACK and to order the extracted data to determine new instances of ENTITY_CLASS: PULSE. Thus, these files are part of the data bridge between the two ENTITY_CLASSes in TLS.

3.1.10 Summary of Phase 1

This section has outlined a general procedure for defining the elements needed to specify the requirements for the DPS. At this point many of the requirements definitions are gross and tentative. The constructs may have been entered into the ASSM as they were defined, or they may just exist on paper. This choice is up to the user, and depends upon the size and nature of the DPS problem, and the number of people involved.

The following "top down" sequence of steps has proved to be the most effective:

1) Define the subsystems relevant to the DPS.

2) Define the interfaces connecting the DPS to the subsystems.

3) Establish the messages passing through the interfaces, and define their data contents.

4) Develop the R_NETs originating at input interfaces.

5) Construct processing steps, tracing backward from the output interfaces when necessary.

6) Define the entities of concern to the DPS, and the associated data hierarchies.

7) Define independent files as necessary.

8) Refine the R_NETs and create new ones, as needed, to link the input and output interfaces.

Some variation on this sequence may be appropriate to certain problems, but the basic principle is to move from known interfaces to internal processing steps, using the data definitions as a vehicle.

The ALPHAs defined at this point will be primitive, and will reflect high level concepts of the nature of the processing. In Phase 3 (Section 3.3) they will be modified, and expanded into subnets, as necessary. But first, Phase 2 (Section 3.2) is needed to consolidate the information developed to this point and to ensure that it forms a consistent basis for detailed development.

3.2 PHASE 2 - EVALUATION OF THE KERNEL

Before completing the definition of the functional requirements structure, it is prudent to enter the information derived in Phase 1 into the ASSM and check the results, both manually and with the aid of Requirements Analysis and Data Extraction (RADX) procedures. Therefore, all information from the paper constructs that has not been previously entered should be loaded into the ASSM. This nucleus of information, called the "kernel", constitutes the minimum needed to document the major elements of the functional requirements definition.

3.2.1 Data Naming Conventions

RSL has been designed to force an early precision of thought with respect to data definition. A single item of information may require several different DATA names in the course of its existence in the system. These are applied as the usage of the information and its properties of transience or permanence dictate. Naming is a tedious process, and may lead to a variety of awkward names, but it is mandatory in the development of a coherent, unambiguous DPS model. While the RSL translator will detect most common ambiguities or conflicts, the user should continually refine his understanding of the data flows within the DPS in order to catch more subtle errors.

The entity data are the first to display the constraints imposed by RSL naming conventions. In general, it is necessary that a data element exist in only a single hierarchy; the sole exception is that it may exist in multiple MESSAGEs. Thus, the identifier of an image being tracked is both the TARGET_ID ASSOCIATED WITH PULSE and the IMAGE_ID ASSOCIATED WITH IMAGE. The two values are the same when the TARGET_ID is assigned, but note that the destruction of an instance in one ENTITY_CLASS is unrelated to the existence of an instance of any other. The meaning of a single identifier, if the IMAGE instance were destroyed while the PULSE was still active, would be indeterminate. It is only to resolve such indeterminacy that the naming rules are imposed; here, the "same" information is given different names for the two occurrences in different hierarchies.

The identifier used for an IMAGE is given to the CC_RESPONSE R_NET as a HO_ID (handover identifier). The same name is applied to a drop-track

command when a TERMINATION message is sent, and also any of three MESSAGEs through the DATA_RECORD interface, whenever information about that image is updated. However, when the information is to be held in global storage (such as for an entity), a different name must be applied (IMAGE_ID for the IMAGE class, TARGET_ID for PULSE).

3.2.2 Structural Data Definitions

R_NETs can be entered into the ASSM in two ways. Using REVS in the ONLINE mode, the nets can be entered interactively via the Anagraph terminal at the ARC facility. Using REVS in the OFFLINE mode, the nets can be specified by an input deck of RSL statements. If the OFFLINE mode is used, certain DATA which are integral to the R_NET structure must be defined with appropriate statements before the set of statements which define the STRUCTURE of the R_NET can be integrated. Prior data definition is not necessary if the Anagraph terminal is used, but should be accomplished within Phase 2 for completeness of the kernel.

Structural DATA elements include those which are referenced on the network STRUCTURE in FOR EACH or SELECT statements; CONSIDER statements associated with an OR node; and conditional expressions associated with an OR node. The latter two types are called "selection variables." Other DATA which should be defined at this point are those DATA which DELAYS the occurrence of an EVENT or ORDERS a FILE.

The FOR EACH statement may be absolute or conditional. The object upon which the statement operates can be an ENTITY_CLASS, ENTITY_TYPE, or FILE. While the FILE is usually associated with one of the interface or entity hierarchies, assumed to be defined previously, some FILEs may exist independently from any hierarchy, as discussed in 3.1.9. It should be verified that all elements used in FOR EACH and SELECT statements are declared in the ASSM prior to entry of R_NET definitions which use them. Elements used in the condition part of a conditional FOR EACH must be defined as discussed below for selection variables.

Selection variables require added definition at this point because of the mechanics of the RSL translation process. In addition to declaration of the DATA name, its TYPE must be identified. Further, if the TYPE is ENUMERATION, the RANGE of values must be defined before the R_NET STRUCTURE can be translated.

While not mandatory, it is recommended that the LOCALITY and USE attributes for selection variables also be defined at this point, and be verified manually because REVS consistency analysis does not include DATA referenced in conditional expressions. Thus, the software cannot detect definition errors until data flow analysis or SIMGEN executions. Figure 3-11 shows RSL inputs which are typical TLS examples of selection variable definition.

```
DATA: MODE,
      LOCALITY: GLOBAL,
      RANGE: "ENGAGED,STANDBY",
      TYPE: ENUMERATION,
      USE: BOTH,
      OUTPUT FROM:

DATA: LAST_PULSE,
      LOCALITY: GLOBAL,
      TYPE: REAL,
      USE: BOTH,

DATA: TRACK_RATE,
      LOCALITY: GLOBAL,
      TYPE: REAL,
      USE: BOTH,

DATA: TEMP,
      LOCALITY: LOCAL,
      TYPE: REAL,
      USE: BOTH,
```

Figure 3-11 RSL Data Entry

The identification of selection variables within a system is usually straightforward. These are simply the decision parameters whose value determines whether one series of processing steps or an alternate series is to be executed. When multiple message types pass through an INPUT_INTERFACE, the type identifier contained in the message is, with no known exceptions, a selection variable (e.g., COMMAND_ID in the CC_IN interface hierarchy). Since the data input to an ALPHA is fixed, and since each message type implies either different data content or different processing, there must exist at least one unique ALPHA for each message type.

3.2.3 Entering R_NETs in the ASSM

The required information about each R_NET is its name, enabling mechanism, and structure. In the current version of REVS, names assigned to R_NETs, SUBNETs, and ALPHAs must be unique within the first eight characters. For all other RSL elements, names must be unique over a field of sixty characters, which is the maximum name length allowed.

The enabling mechanism of an R_NET is either a single INPUT_INTERFACE or one or more EVENTs. If a message passing through an interface is the mechanism, then the first statement in the R_NET STRUCTURE is an INPUT_INTERFACE name declaration. In the case of EVENTs, the EVENT name does not appear in the STRUCTURE of the enabled R_NET. However, it appears in the STRUCTURE of the enabling R_NET at the appropriate point.

When the R_NET is entered into the ASSM via card input, data elements which appear in the STRUCTURE must be predefined in the ASSM. These data are discussed in 3.2.2. The R_NET STRUCTURE is discussed in below.

3.2.4 The STRUCTURE of an R_NET

Flows through the system are specified in RSL as Requirements Networks (R_NETs). R_NET flow STRUCTURES consist of nodes, which specify processing operations, and the arcs which connect them. The processing nodes are ALPHAs, which are specifications of functional processing steps, and SUBNETs, which are specifications of processing flows at a lower level in the hierarchy. The processing nodes are single-entry, single-exit.

In addition to the simple sequential flow which may be represented by connecting this type of node, more complex flow situations are expressible in RSL by the use of structured nodes which fan-in and fan-out to specify different processing paths. These nodes are variations of AND and OR nodes, and include an OR with a CONSIDER statement. The REVS Users Manual contains an extensive discussion of these structural elements. Also, the TLS examples in the Appendix show the actual format of various structures.

Indentation is used in the STRUCTURE declaration to facilitate reading. When entering the declaration on cards, using REVS in the OFFLINE mode, the indentation format typified by the REVS output examples in the appendices should be followed. This is not mandatory, but it is strongly recommended

in order to check the output against the input. In this way, the user can quickly detect where REVS has interpreted the STRUCTURE in a different way than intended.

In addition to describing the processing flow of an R_NET, the STRUCTURE declaration may also serve to declare the existence of the named ALPHAs and SUBNETs. The STRUCTURE of any SUBNET should also be declared at this time. Further definition of the attributes of ALPHAs will take place in Phase 3.

If Anagraph or CALCOMP plots of the structures are desired, graphic data must be entered for each R_NET. Graphic coordinates may be entered from the Anagraph terminal or generated by a RADX PLOT command. The Anagraph plots are useful while working interactively at the terminal. The CALCOMP plots are more useful for permanent retention and for documentation, as shown in Appendix G.

3.2.5 Checking the Kernel with the Aid of RADX

As part of the auditing process, it is useful here to define a set of RADX directives which assist in determining the completeness of entries into the ASSM. Two operations are recommended which are simple, but revealing:

1) APPEND R_NET STRUCTURE, ENABLED.
   LIST R_NET.

2) APPEND SUBNET STRUCTURE, REFERRED.
   LIST SUBNET.

These directives generate a listing of the R_NETs with their structures and enabling events or interfaces. Since each R_NET requires a structure, and an enabling condition, any failure of that class is detectable here. Similarly, the correctness (agreement with intention) of EVENT naming may be confirmed.

While it is possible to confirm the structures by inspection of the RADX output, reading the equivalent diagrams (generated by RNETGEN) is generally simpler; since they are equivalent representations, either the CALCOMP illustration or the RADX listing may be employed for the purpose.

(Note that viewing the Anagraph output at the terminal is less useful since all element names are truncated to three characters and since branching criteria are not immediately displayed.)

It is in data analysis that the RADX tools are most useful at this stage. Defining two hierarchies for data output:

HIERARCHY FILES = FILE CONTAINS DATA DATA INCLUDES DATA.

HIERARCHY DATUM = DATA INCLUDES DATA.

It is desired to identify the DATA items and the FILEs which are not linked with higher data levels (entities, or messages). One way derives from the following definitions:

SET A = ALL WITHOUT ASSOCIATED.

SET B = A WITHOUT MAKES.

SET C = B WITHOUT CONTAINED.

SET D = C WITHOUT INCLUDED.

Now the RCL (RADX) directive: LIST B WITH HIERARCHY FILES will generate both the file names and the data comprising each such file where the file is not associated with an entity and does not make a message. Such a free-standing file should be a conscious product of requirements engineering and not an accident.

Free-standing DATA may be extracted by: LIST D WITH HIERARCHY DATUM. Note that the HIERARCHY DATUM is used here as a convenience to limit the output to the names of the top data level not constituting part of a hierarchy; it does not in fact cause INCLUDED DATA to be output, since the SET from which the extraction is effected (D) has no members which are INCLUDED in any DATA. To complete tracing of the data hierarchies, yet another SET would have to be defined containing the DATA extracted by LIST D WITH HIERARCHY DATUM, and that SET would then be LISTed with HIERARCHY DATUM.

Typically, free-standing DATA and FILES may be local or global constants. Each item extracted by the methods outlined above should be assessed to determine whether in fact it should be isolated or is properly a constituent of an entity or interface hierarchy.

Finally, the following HIERARCHies may be defined, and each may be used to complete the RADX directive: LIST ALL WITH HIERARCHY _____.

```
HIERARCHY ENTITY =
    ENTITY_CLASS ASSOCIATES DATA
    ENTITY_CLASS ASSOCIATES FILE
    ENTITY_CLASS COMPOSED ENTITY_TYPE
        ENTITY_TYPE ASSOCIATES DATA
        ENTITY_TYPE ASSOCIATES FILE
          FILE CONTAINS DATA
            DATA INCLUDES DATA.

HIERARCHY INFACE =
    INPUT_INTERFACE PASSES MESSAGE
        MESSAGE MADE DATA
        MESSAGE MADE FILE
          FILE CONTAINS DATA
            DATA INCLUDES DATA.

HIERARCHY OUTFACE =
    OUTPUT_INTERFACE PASSES MESSAGE
        MESSAGE MADE DATA
        MESSAGE MADE FILE
          FILE CONTAINS DATA
            DATA INCLUDES DATA.
```

The result is to extract from the ASSM the complete kernel of the requirements, as illustrated in Appendix F.

3.2.6 Summary of Phase 2

This phase has been a period of consolidation, between the initial definitions of Phase 1, and the completion of those definitions in Phase 3. Nonetheless, the end of Phase 2 is an important milestone in the total effort.

For the first time, the kernel of the requirements construct has been loaded into the ASSM as a whole. Many of the early mistakes and inconsistencies will be detected by the RSL translator during the entry process. More important, the requirements engineer can now use the facilities of the Requirements Analysis and Data Extractor (RADX).

The introductory uses of RADX, in 3.2.5, will be built upon and expanded in Phase 3. Eventually, each user will probably compile his own library of RADX procedures. In future efforts these "off-the-shelf" procedures can be used as part of the overall evaluation sequence.

3.3 PHASE 3 - COMPLETION OF THE FUNCTIONAL DEFINITION

The work through Phase 2 has provided definition of the higher-level elements of data hierarchies; definition of R_NETs, their STRUCTUREs, and their enabling events; and declaration of the existence of ALPHAs and their place in the R_NETs. This information has been entered into the ASSM and has been subjected to various forms of automated analysis and review.

It now remains to complete the definition of these elements to provide a basis for construction of an executable simulation. Each ALPHA must be defined in terms of its data transactions, and definition of all DATA which are known to be INPUT TO or OUTPUT FROM an ALPHA must be completed. In general, the DATA which can be described at this stage are either those global to multiple R_NETs, or those local DATA which make messages. The need for additional local data internal to the R_NETs will be discovered in the development of executable descriptions (Section 3.5). At that time the need for definition of lower levels of the existing data hierarchies may be apparent.

During this process, the original primitive ALPHAs will often need redefinition. Additional ALPHAs can be added, or the original ALPHAs can be redefined as SUBNETs, which preserves traceability and minimizes modification of the R_NETs. Occasionally, the STRUCTURE of the net may change, or new nets may be added, as the processing requirements evolve.

The following phases will be devoted to construction of a functional simulation, using BETAs, which are executable descriptions of the ALPHAs. It should be emphasized that this activity develops and executes a simulation of the requirements for the DPS software, not of the software itself. The primary goal of this simulation is to ensure that the functional definition of requirements are complete and consistent.

Since the major interactions of the R_NETs have been defined in the previous phases (Sections 3.1 and 3.2), it is now possible to partition the work into essentially isolated efforts for several engineers. Each parcel should consist of one or more R_NETs, each considered by a single engineer or group. Since the relationships among R_NETs have already been defined, the interactions among parcels will be restricted to joint agreement on the content of communications (usually individual DATA) between

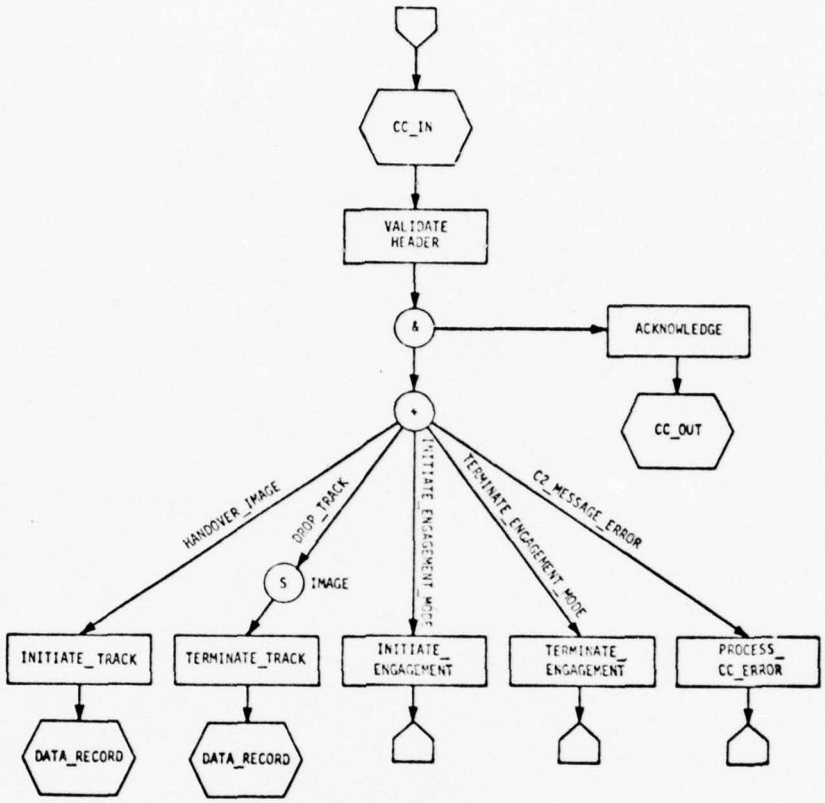
pairs of engineers, and need not be coordinated extensively over the system as a whole. The sole exception to that rule is in naming conventions, where a possibility of conflict exists, and where control is useful.

3.3.1 Entity Transitions

Whenever an instance of an ENTITY_CLASS is selected, created, or destroyed, the context of the processing flow is altered significantly. To clarify the flow, each such operation is represented as a node on a STRUCTURE (R_NET or SUBNET) or as a relationship on an ALPHA. The SELECT node is always qualified by the conditional which defines the instance to which subsequent processing refers; the selection persists until another SELECT or FOR EACH node is encountered; note that each branch of processing is assumed to be completed before another is entered, so that the SELECTion of an instance of any ENTITY_CLASS need be performed only once on a branch (in most cases).

It must be remembered in defining the nodes and relationships which generate entity transitions (i.e., SELECT, FOR EACH, CREATES, DESTROYS, and SETS) that each is interpreted to mean that the transition is <u>always</u> effected when the node is reached. Thus, an ALPHA which SETS an ENTITY_TYPE will always do so. Where the interpretation is the same for such a relationship as for, say, INPUTS, REVS enforces it even more strongly. That is, REVS will report on the error (but will build a simulator using the data) when an undeclared DATA input is used in a BETA or GAMMA; however, REVS will provide a transition of ENTITY_TYPE if and only if the SETS relationship is declared. The transition will be for exactly the instance selected at the time the related ALPHA is encountered on the net.

The R_NET for CC_RESPONSE developed in Section 3.1 is illustrated again in this section with entity operations expressed on the net. Comparison with the figures in Section 3.1.6 will show the added detail and information provided by the added nodes. After modifying the structures to reflect the entity transitions, analysis of the kernel (Section 3.2.5) should be repeated.



3.3.2 Data Transactions

There was an initial concept of the DATA which were to be INPUT TO and OUTPUT FROM each ALPHA when the ALPHA was placed in the STRUCTURE of an R_NET. Now the preliminary gross concept must be precisely defined. During this process, additional data definitions or more detailed definition of existing hierarchies may be necessary.

As the data transactions and the hierarchy transitions discussed in Section 3.3.4 are developed, the processing steps within each ALPHA which transform the inputs into the required outputs must be considered. The conceptual ideas of the procedure may be noted in the ASSM, in English text, or using the RSL DESCRIPTION attribute. A structured English description of the processing can later be used for reference during development of the BETAs which are written in PASCAL. If, at this point, it becomes apparent that significant logical branching must occur within the ALPHA, the ALPHA

can be redefined as a SUBNET. In this manner the logical structure requirements are made explicit in the STRUCTURE definition of the SUBNET and new ALPHAs are defined to specify the operations within this structure.

As an example of the thought process involved with data transactions, consider the ALPHA named INITIATE_TRACK (later renamed TRACK_INITIATE). This ALPHA is on the CC_RESPONSE R_NET and is on the processing path activated by a HANDOVER message of type HANDOVER_IMAGE (=COMMAND_ID). Since COMMAND_ID was used as a selection variable in order to reach the ALPHA, it is unlikely that it will be used within the ALPHA. However, the remaining data content of the MESSAGE must be used within the ALPHA because there are no other ALPHAs on this path which could use the unique content of this message type. These DATA are HO_ID, INITIAL_STATE, and INITIAL_ COVARIANCE. Thus, these DATA will be defined as INPUT TO the ALPHA.

.TRACK_INITIATE must FORM a MESSAGE to be PASSED BY the OUTPUT_INTERFACE named DATA_RECORD. It is clear that TRACK_INITIATION is the proper message to be formed and passed. Thus, the DATA in that message must be OUTPUT FROM the ALPHA. These are HO_ID, INITIAL_STATE, and TIME_OF_INITIATION. The first two DATA are present in the inputs, but the third is not.

The remaining DATA item for the MESSAGE is the time of arrival of the incoming command at CC_IN. REVS maintains a DATA item called CLOCK_ TIME which is set to the simulator time at the initiation of each R_NET, and which is not to be altered by any processing. (Since time is not advanced during the "execution" of an R_NET, CLOCK_TIME is not updated along a net.) Note that CLOCK_TIME is a DATA item predefined within the RSL NUCLEUS in the ASSM: a second DATA item is maintained by REVS, FOUND, which is TRUE if the last SELECT operation retrieved an instance satisfying the selection criterion. Neither element may ever be OUTPUT by an ALPHA. Either element may be INPUT TO an ALPHA, as CLOCK_TIME is INPUT TO: TRACK_INITIATE for assignment as TIME_OF_INITIATION, which is then OUTPUT for the MESSAGE.

What additional processing is needed in TRACK_INITIATE? Its primary function is to use the data provided in the HANDOVER message to initiate track on a new image. Thus, it is logical to assume that the ALPHA CREATES a new instance of ENTITY_CLASS: IMAGE, and designates it as ENTITY_TYPE:

IMAGE_IN_TRACK. Hence, the ALPHA should output the DATA associated with this ENTITY_CLASS and ENTITY_TYPE. These are: IMAGE_ID, ENTRY_TIME, STATE, COVARIANCE, TRACK RATE, WAVEFORM, and LAST_PULSE.

It is also reasonable to assume that TRACK_INITIATE should assign the values of HO_ID, CLOCK_TIME, INITIAL_STATE, and INITIAL_COVARIANCE to IMAGE_ID, ENTRY_TIME, STATE, and COVARIANCE, respectively. These assignments are unconditional, as no choice criteria are indicated in the specifications. For the moment, it is sufficient to defer the question as to the origin of TRACK_RATE, WAVEFORM, and LAST_PULSE, and simply to define them as OUTPUT FROM the ALPHA. Figure 3-12 shows the declaration, in RSL, of all the data transactions of TRACK_INITIATE. Note that a DATA item, DATA_RECORD_TYPE is defined for the output message consistent with MESSAGE definition requirements discussed in Section 3.1.3. From our concept of the processing, it appears that the original ALPHA is adequate and no additional ALPHAs, or redefinition as a SUBNET, are required.

```
ALPHA:  TRACK_INITIATE,
  INPUTS:
       DATA:  CLOCK_TIME
       DATA:  HO_ID
       DATA:  INITIAL_COVARIANCE
       DATA:  INITIAL_STATE,
  OUTPUTS:
       DATA:  COVARIANCE
       DATA:  DATA_RECORD_TYPE
       DATA:  ENTRY_TIME
       DATA:  IMAGE_ID
       DATA:  STATE
       DATA:  TIME_OF_INITIATION
       DATA:  TRACK_RATE
       DATA:  WAVEFORM,
```

Figure 3-12 RSL Initial ALPHA Entry

3.3.3 RADX Evaluation of Data Transactions

In addition to confirming entry of attributes and relationships through RADX directives (e.g., LIST ALPHA.), it is now useful to extract the specific cases which are potential errors, or are at least anomalous to the point where they demand special attention. For example, it is possible for an

ALPHA either to have no INPUTS or to have no OUTPUTS, or even to have neither, without its being erroneous; but the normal case is that each ALPHA will have both. Having completed the RSL declarations about the ALPHAs at this stage, it is meaningful to define the following SETS for RADX analysis of the ALPHA definitions existing in the ASSM at this point.

1) SET A = ALPHA WITHOUT INPUTS.
2) SET B = ALPHA WITHOUT OUTPUTS.
3) SET C = A WITHOUT OUTPUTS.

Remembering that the declaration of a SET generates a count of its members, it may be discovered that SET C is empty; this would indicate that each ALPHA has either an INPUTS or an OUTPUTS relationship (or both). However, in TLS, several ALPHAs have neither. In particular, the error-processing elements (C2_ERROR_PROCESSING) are required to exist, but have no other specifications; therefore, they have no accesses defined. Similarly, the ALPHA: ACKNOWLEDGE exists solely so that the ACKNOWLEDGEMENT message may be FORMED; since the processing modifies no DATA, the ALPHA has neither INPUTS nor OUTPUTS.

In general, an ALPHA without INPUTS is one which operates on the system as a whole, rather than on information retained by the DPS. For example, ENGAGEMENT_INITIATION and TERM_ENGAGEMENT accomplish their purposes by the occurrence of the appropriate message; only the value of the selection variable (COMMAND_ID) is required to initiate them, and these ALPHAs execute independently of the state of the global data base.

In general, an ALPHA with INPUTS but without OUTPUTS is a signal of a specification problem. Trivially, the only reason for acquisition of DATA for processing is that it may effect some generation of DATA for OUTPUT. To construct a valid case for an ALPHA which must accept INPUTS but is not required to provide OUTPUTs has not yet been demonstrated.

3.3.4 Hierarchy Transitions

In addition to the elementary data relationships, each ALPHA may modify DATA within hierarchies in a variety of ways. Given the engineering concept of the action of the ALPHA, it is possible to express its action on the hierarchies in terms of the RSL directives: CREATES, DESTROYS, SETS, and FORMS.

There are two fundamental generative operations that an ALPHA may express: CREATES and FORMS. They declare the need for a new instance of a named ENTITY_CLASS, or for a named MESSAGE, respectively. In response to either directive it is required that the specified INITIAL_VALUEs for all associated DATA with such values be assigned. This is done automatically when REVS acts upon an INITIAL_VALUE definition. If an INITIAL_VALUE is not defined, REVS will assign a default value according to the TYPE of the DATA. These values are (0, 0.0, FALSE, first defined value in the RANGE) for the respective types (INTEGER, REAL, BOOLEAN, ENUMERATION). Once the DATA associated with the instance are initialized, they may be assigned current values by assignment statements within the BETA.

When an instance of an ENTITY_CLASS is CREATED, the ALPHA which CREATES the instance normally SETS the ENTITY_TYPE. Otherwise, only the DATA and FILEs common to all types in the class can be defined. As the instance persists in the system, its type will evolve (i.e., an ALPHA SETS it to a different type). The ALPHAs in which such changes are effected are normally obvious from the R_NET. In each case, such an ALPHA SETS ENTITY_TYPE to the appropriate ENTITY_TYPE name.

For example, the discussion, in 3.3.2, of the ALPHA named TRACK_INITIATE revealed that the ALPHA performs all of the actions above. It FORMS the MESSAGE named TRACK_INITIATION. Also, it first CREATES an instance of ENTITY_CLASS: IMAGE, then SETS the instance to ENTITY_TYPE: IMAGE_IN_TRACK. Figure 3-13 shows how these declarations are written in RSL.

```
ALPHA:  TRACK_INITIATE,
  CREATES:
       ENTITY_CLASS:  IMAGE.
  FORMS:
       MESSAGE:  TRACK_INITIATION.
  SETS:
       ENTITY_TYPE:  IMAGE_IN_TRACK.
```

Figure 3-13 RSL Addtional ALPHA Entry

When there is no further need for the DPS to retain data related to a specific instance of an ENTITY_CLASS, or to be aware of the existence of the instance in the external world, the "instance of the ENTITY_CLASS" can be DESTROYED BY an ALPHA. The disposition of a PULSE in TLS depends on whether it is a LOST_PULSE or a RETURNED_PULSE. For a LOST_PULSE, the instance is DESTROYED (no longer of value) after it has been accounted for in the ALPHA named SET_LOST. For a RETURNED_PULSE, the instance is not destroyed until it has also been accounted for in the ALPHA named SET_PULSE. Note that, in this case, the instance of ENTITY_CLASS: PULSE can be DESTROYED in either way, but not before it has been accounted for in both. This is because the R_NETs execute independently of one another, and no specific sequence is otherwise required.

3.3.5 Further Data Definition

Through the previous work many, if not most, of the DATA in the system have been named, and their relationships with ALPHAs have been established. In order to complete the requirements specification and construct an executable simulation, however, the "attributes" of the DATA must be defined. The three principal attributes are LOCALITY, TYPE, and USE.

3.3.5.1 Locality

DATA and FILEs may have different required accessibility in the system. The range of accessibility of an item is denoted by the attribute LOCALITY, which may have values of LOCAL or GLOBAL. Items of DATA or FILEs which are LOCAL are associated with the R_NETs in which they are used and are unknown outside of these R_NETs. LOCAL DATA exist only during the invocation of the R_NET to which they are LOCAL. They are created when the flow token is generated at R_NET ENABLEment and cease to exist when the flow token leaves that R_NET. ALPHAs which use LOCAL DATA and FILEs may appear on more than one R_NET; therefore, it is possible for a single DATA item or a FILE to be LOCAL to more than one R_NET. However, these are different instances of the DATA or FILE which have no relation to each other; each has a completely separate existence, controlled by the R_NET in question. Note that data local to an R_NET are given their INITIAL_VALUEs at initiation of that net.

GLOBAL DATA and FILES are accessible by more than one R_NET and exist over more than one R_NET invocation. DATA and FILEs which are ASSOCIATED with an ENTITY_TYPE or an ENTITY_CLASS are tied to the entity instances to which they belong. They are created when the instance is CREATED and persist until the instance is DESTROYED. Items which are not ASSOCIATED with entities are permanently housed in the global data base, and may exist throughout the duration of the system.

Thus, DATA and FILEs which belong to an interface data hierarchy (e.g., MESSAGE) are implicitly LOCAL and required by REVS to be LOCAL. DATA and FILEs associated with an entity data hierarchy are implicitly GLOBAL and required by REVS to be GLOBAL. DATA and FILEs not associated with either type of hierarchy have not implicit locality. Even if the locality is not explicitly defined, an item exists in the data base at all times, accessible to the R_NETs, but in an "undefined" state. Failure to declare LOCALITY can produce weird consequences later, if the element involved has not implicit locality. On the other hand, declarations are not needed if the locality is implicit. At best they are redundant, and at worst they are misleading because REVS overrides, during simulation generation (SIMGEN), any declaration in the ASSM which conflicts with the implicit locality.

While it is possible to manually identify which DATA and FILEs need LOCALITY declarations, and which do not, it is easier and less risky to use a RADX procedure to do this. One such procedure is discussed in 3.3.6.

For the remaining items which need a LOCALITY declaration, the source of the item should be considered. If an independent DATA item or FILE is not OUTPUT FROM some ALPHA on an R_NET previous to its being INPUT TO some other ALPHA on that net, it cannot be LOCAL to that R_NET. If the item is OUTPUT FROM an ALPHA on some other R_NET, the item is GLOBAL unless an error has been made in the INPUTS and OUTPUTS definitions. In many cases the correct locality is obvious. The inability to find a source for an independent item on any of the R_NETs indicates a specification deficiency.

If a DATA item or FILE is OUTPUT FROM an ALPHA on a net other than that which uses the item as input, the indicated LOCALITY is GLOBAL. If no other R_NET generates the item, then the origin of the item is on a previous execution of the R_NET which uses it.

If the ALPHA which OUTPUTS an item does not clearly precede the ALPHA which INPUTS it, care must be taken to ensure that the item has an initial value. The default values assigned by REVS in the absence of an INITIAL_VALUE declaration were described in 3.3.4. If these are unacceptable, the user must declare the correct INITIAL_VALUE. Note that "clearly precede" in the context above means that the ALPHA which OUTPUTS the item is either: 1) before the ALPHA which INPUTS it, on the same path within an R_NET, or 2) precedes the ALPHA which INPUTS the item on a single path linked by enabling EVENTs (generally without DELAYS).

3.3.5.2 Type and Range

Other details about DATA must be known for purposes of simulation. BETAs and GAMMAs are executable code which are meaningful only if more is known about the DATA than should be stated as a requirement. In addition, a hierarchy of DATA may be stated as a requirement, but a functional simulation (using BETAs) may employ DATA only part way down the hierarchy. That is, the simulation may use one DATA item to represent a part of an entire hierarchy. The characteristics of this summarized DATA must be stated for the

simulation to execute using the RSL USE attribute. Since an analytic emulation (using GAMMAs) might not employ the same DATA, the type information for BETAs may be different from that for GAMMAs. The only DATA that can be used in the BETAs and GAMMAs and have their values communicated between ALPHAs, however, are those whose TYPE has been defined.

The attribute TYPE contains the necessary information for typing of the DATA. This attribute may have values REAL, INTEGER, BOOLEAN, or ENUMERATION. A DATA item with type ENUMERATION corresponds closely to one with a scalar type in PASCAL; that is, it has values which are denoted by identifiers. The legal values for ENUMERATION types are given in the RANGE attribute.

The TYPE declared need not correspond to that of the actual data in the real DPS. Rather, it should be chosen to reflect the purposes of the functional and analytic simulations, and the fidelity required in those simulations. For instance, in the real DPS, unique alphanumeric text strings may be used to identify objects. If these are an open set, they cannot be represented by a DATA element with TYPE: ENUMERATION because that defines a closed set. However, for purposes of simulation, the DPS property of interest is the ability of the DPS to distinguish between unique identifiers. Thus, by defining the identifier as TYPE: INTEGER, and representing each object or class of objects by a unique integer value, a representation adequate for the purposes is obtained.

Similarly, in TLS, identifiers such as HO_ID, IMAGE_ID, and RADAR_ORDER_ID are represented by integers. In the real TLS some symbolic code convention would probably be used. In like fashion, message identifiers such as COMMAND_ID which form a closed set are represented as DATA with TYPE: ENUMERATION. The values defined are for explanatory purposes. The values in the actual TLS would certainly be different.

Since the appropriate TYPE for DATA is strongly dependent on its USE, the choice may be deferred until the USE attribute has been determined.

3.3.5.3 Use

Further qualification of the use of a DATA item in the simulation is given by the attribute USE. The value of this attribute may be BETA, GAMMA, or BOTH denoting that the data item is the lowest level in the data hierarchy which will be used in the corresponding simulation.

Frequently, even the first declaration of a DATA item makes clear its USE in the system. For example, if the element is known to correspond to a single unit of information (bit, byte, or word) in implementable software, then it is probably used in BOTH beta and gamma models. Normally, a selection variable will be in this class. Similarly, it is likely that an item which INCLUDEs others in the detailed modeling, but which may be treated as a whole for functional modeling will have USE:BETA. It is unlikely that any element with USE restricted to GAMMA will be recognized at this stage of development, since its declaration would be primarily in support of analytic simulation; the exception would occur if a highly detailed interface specification gave low-level DATA definitions, for which higher levels would suffice in a functional model.

For example, in the TLS definition, INITIAL_STATE is a single DATA element with USE:BETA. In the external world "state" is defined by a position vector, a velocity vector, and perhaps an acceleration vector, each with three components. Such detail is not needed for the functional simulation. However, an analytic simulation has need for these elements. Thus, eventually each of the components will be defined to have USE: GAMMA. Similarly, for the functional simulation, INITIAL_COVARIANCE can be represented by a single element with USE:BETA. Later, its matrix components can be defined with USE:GAMMA when they are needed for analytic simulation.

3.3.5.4 Values

Values are the ultimate object of defining DATA. At the lowest level in a hierarchy of DATA the requirements engineer may specify the attributes UNITS, MAXIMUM_VALUE, MINIMUM_VALUE, INITIAL_VALUE, and RESOLUTION. The attribute UNITS is given separately from the various types of values, both to maintain consistency with their specification and to enable requirements engineers to indicate the minimum possible information about a DATA's value,

its UNITS. In nearly all cases it will be known the units are milliseconds or microseconds even if the specific value has not yet been determined. Separating UNITS from the values enables phased specification of the best information that is available at the initiation of DATA item definition.

Since the attributes UNITS, MAXIMUM_VALUE, MINIMUM_VALUE, and INITIAL_VALUE are not vectors, the definition of different UNITS and values of a DATA set above the lowest level in the hierarchy is not possible. RESOLUTION describes the required maximum value of the least significant bit for the DATA in units described in the UNITS attribute.

3.3.6 Evaluation of the ASSM Using RADX

The Requirements Analysis and Data Extraction (RADX) function of REVS is the tool used to investigate the state of the Abstract System Semantic Model (ASSM). RADX provides commands that allow the performance of several functions:

- Identification and listing of elements in the ASSM that do or do not meet some criterion.
- Listing of ASSM elements in such a manner as to be suitable for inclusion in requirements documents.
- Listing of RSL element, attribute, and relation definitions.
- Analysis of the ASSM to identify requirements that are ambiguous or inconsistent.

RADX can be used to extract specific data for analysis and to detect inconsistencies and omissions in the ASSM. This capability is also available to management to evaluate progress of the development activity, and to independent analysts who may be assigned to verify the results of the requirements engineer.

The following subparagraphs discuss typical uses of RADX in evaluating the work done in Phase 3, and in identification of work remaining to be accomplished. These examples are not meant to be comprehensive, nor are they the only way to do the task. A comprehensive catalog of RADX procedures would be a very thick document; also, there are many ways to do a specific extraction task effectively. Other examples can be found in the REVS Users Manual. The purposes here are to suggest some of the possible uses of RADX, and to encourage further creativity with the REVS facilities.

3.3.6.1 RADX Evaluation of Data Origin and Usage

A DATA item may exist in the relational data base in any of the following ways:

1) OUTPUT BY an ALPHA;

2) INCLUDED IN a DATA OUTPUT BY an ALPHA;

3) CONTAINED IN A FILE OUTPUT BY an ALPHA; or

4) MAKE a MESSAGE which is PASSED BY an INPUT_INTERFACE.

Included in the last category are DATA INCLUDED in DATA which MAKES such a MESSAGE and DATA CONTAINED IN a FILE which MAKES such a MESSAGE.

RADX is used to extract exceptions to the above rules, since they constitute apparent cases of 'creative memory' -- data extractable from the global data base which never need to be entered. A legitimate case of creative memory is a constant with a defined INITIAL_VALUE; in fact, DATA constants are properly defined in just such a manner. But any non-constant data item which fits in none of the above categories is an apparent error, and worthy of detailed analysis. (Note that a DATA element which is never INPUT TO an ALPHA and which does not MAKE a MESSAGE which is PASSED BY an OUTPUT_INTERFACE is also worthy of scrutiny, and may be similarly analyzed at this stage. Among the DATA which will properly be detected at this step are those REFERRED by an R_NET-- e.g., selection variables.)

With the aid of the hierarchy definitions in 3.2.4, RADX procedures can be constructed to isolate the DATA described above. First RADX commands, which are used in both cases, are defined as:

SET INALPH = DATA THAT IS INPUT BY ALPHA.
SET OUTALPH = DATA THAT IS OUTPUT BY ALPHA.

The additional commands below will provide a listing of DATA INPUT TO an ALPHA which does not have an INITIAL_VALUE, and is not in a MESSAGE PASSED BY an INPUT_INTERFACE, and which is not OUTPUT FROM some ALPHA on some R_NET.

```
SET INMSG = ALL IN HIERARCHY INFACE.
SET INDATA = INMSG AND DATA.
SET INOUT = INALPH MINUS OUTALPH.
SET INOUT = INOUT MINUS INDATA.
SET INOUT = INOUT WITHOUT INITIAL_VALUE.
LIST INOUT WITH HIERARCHY DATUM.
```

If, instead, the commands shown below are used, a listing of DATA which is OUTPUT from an ALPHA and not INPUT TO an ALPHA and not in a MESSAGE PASSED BY an OUTPUT_INTERFACE will be generated.

```
SET OUTMSG = ALL IN HIERARCHY OUTFACE.
SET OUTDATA = OUTMSG AND DATA.
SET OUTIN = OUTALPH MINUS INALPH.
SET OUTIN = OUTIN MINUS OUTDATA.
LIST OUTIN WITH HIERARCHY DATUM.
```

3.3.6.2 RADX Evaluation of File Activity

In 3.3.2 the interpretations when a FILE is INPUT TO and/or OUTPUT FROM an ALPHA were presented. RADX commands can be used to isolate various INPUT combinations for verification. The following are useful examples:

```
SET A = FILE WITH INPUT.
SET B = FILE WITH OUTPUT.
SET C = A OR B.
SET D = FILE MINUS C.
SET E = B MINUS A.
SET F = A MINUS B.
SET G = A AND B.
```

Set D consists of FILEs which are neither INPUT TO nor OUTPUT FROM an ALPHA.

Set E consists of FILEs which are only OUTPUT FROM some ALPHA. Given that additions were made to the FILE for some purposes, then the only valid purpose, other than for INPUT TO some ALPHA, is for items which MAKE an output MESSAGE. The following RADX commands provide a listing of the remaining which do not MAKE an output MESSAGE and which should be checked for errors.

```
SET H = E MINUS OUTMSG.
LIST H.
```

Set F consists of FILEs which are only INPUT TO some ALPHA. This indicates that the FILE instances are accessed, and possibly deleted, within the ALPHA, i.e., the BETA or GAMMA description. Since the FILE is not OUTPUT FROM an ALPHA, it cannot originate within the DPS. Therefore, it must appear in an input MESSAGE. The following RADX commands provide a listing of the remaining items which do not MAKE an input MESSAGE and which should be checked for errors.

```
SET I = F MINUS INMSG.
LIST I.
```

Set G consists of FILEs which are both INPUT TO and OUTPUT FROM some ALPHAs, not necessarily the same ALPHA. These FILEs, together with the ALPHAs which operate on them, can be extracted by the following commands.

```
APPEND FILE INPUT, OUTPUT.
LIST G.
```

This listing should be examined to ensure that the FILEs are operated upon as intended. Particular attention should be given to ALPHAs which INPUT and OUTPUT the same FILE. It is these ALPHAs which either modify the instances within a FILE, or add new instances when an appropriate one cannot be found.

3.3.6.3 RADX Evaluation of Entity Activity

It is also useful to verify that ENTITY_CLASSes and ENTITY_TYPEs are manipulated appropriately within the system. Each ENTITY_CLASS must be CREATED BY some ALPHA. Each ENTITY_CLASS should be DESTROYED BY some ALPHA. If it is not destroyed, there must be a valid reason for retaining the ASSOCIATED DATA or FILEs. Further, each ENTITY_TYPE within an ENTITY_CLASS must be SET BY some ALPHA. The members of the following sets represent apparent deviations from these rules and should be examined.

```
SET A = ENTITY_CLASS WITHOUT CREATED.
SET B = ENTITY_CLASS WITHOUT DESTROYED.
SET C = ENTITY_TYPE WITHOUT SET.
```

The following hierarchy is useful to determine the actions on each ENTITY_CLASS in the system.

```
HIERARCHY ENTITY_CHECK =
ENTITY_CLASS CREATED BY ALPHA
ENTITY_CLASS COMPOSED ENTITY_TYPE
    ENTITY_TYPE SET BY ALPHA
ENTITY_CLASS DESTROYED BY ALPHA.
```

The following RADX command will provide a structured listing of each instance of the hierarchy suitable for further analysis.

```
LIST ALL WITH HIERARCHY ENTITY_CHECK.
```

3.3.6.4 RADX Evaluation of Data Attributes

Data extraction can be of major support in development of the executable description, used in BETAs and GAMMAs, although it is not a major contributor to assessment of the result. The fact that RADX cannot penetrate the contents of a BETA or GAMMA to determine its consistency with declarations of relationships and attributes is of little significance, since the consistency and completeness are thoroughly analyzable with the static analyzers, and since the crucial test of simulation generation is then executable.

One of the key operations at this stage of specification development is defining the LOCALITY, USE, and TYPE of all DATA required for functional simulation. LOCALITY is the easiest of the attributes to specify:

1) Using the definitions of 3.2, LIST ALL WITH HIERARCHY ENTITY generates a collection of GLOBAL DATA;
2) Using these definitions, LIST ALL WITH HIERARCHY INFACE and LIST ALL WITH HIERARCHY OUTFACE generates a collection of LOCAL DATA;
3) LIST B WITH HIERARCHY FILES generates the collection of DATA CONTAINED IN FILEs which are not linked to higher levels. Since each such file is inherently either local or global in scope, all CONTAINED DATA have the same LOCALITY as their parent; and
4) LIST D WITH HIERARCHY DATUM identifies the top level of DATA which are not linked into higher levels. Each such item is then assigned a LOCALITY, which is also the value assigned to any DATA item INCLUDED in that top level.

Since the system provides an override of any LOCALITY declaration for ASSOCIATED DATA or those which MAKEs a MESSAGE, any LOCALITY provided by the user would be at least redundant, and at worst misleading, however, REVS will identify these conditions during SIMGEN by issuing Error Messages. Therefore, it is recommended that each DATA item and FILE generated by (1) and (2) above be checked to ensure that no LOCALITY is declared. Similarly, it is best to declare the LOCALITY for each FILE derived from (3), and then to omit LOCALITY for each FILE derived from (3), and then to omit LOCALITY for each DATA item obtained. Finally, each DATA item obtained in (4) is assigned a LOCALITY, and the same LOCALITY is assigned to each DATA item included in it.

Data extraction also supports assigning USE to the DATA items. Define SET A = DATA WITHOUT INCLUDES. Each item in that set must have USE: GAMMA or BOTH. It is given USE BOTH exactly if it is to be a part of the BETA model for some ALPHA. For each item with USE:BOTH, no DATA above it in the hierarchy can have a USE assigned. Note that USE:BOTH and USE:GAMMA may be applied only to the lowest level of DATA defined. Once it is determined that a given lowest-level DATA item will not be included in the functional model, an item above it in the hierarchy must be assigned USE:BETA. That item may be anywhere above the one with USE:GAMMA, except that it cannot also INCLUDE (either directly or through a chain of INCLUDES) any element with USE:BOTH.

By defining a SET A = DATA WITH USE, the engineer determines the items requiring TYPE. It is not mandatory at this stage to TYPE DATA which are to be used only analytically, so that the subset required for functional simulation can be obtained through SET B = A WITHOUT USE = GAMMA. Through examination of the STRUCTUREs and BETAs, the TYPE of each such item is defined, and enumerated DATA are also assigned RANGE.

Note that it is at least misleading, and sometimes likely to induce errors, to define attributes for elements which do not require them. Thus, a LOCALITY:LOCAL declaration for an item ASSOCIATED WITH an ENTITY_TYPE would be overridden by REVS, but would be entered into the ASSM and would appear to the reader of the specification to control the DATA item. Such

a condition would degrade legibility of the document, and should be avoided. The data extractor can be used to determine if any over-specification of this sort has been attempted.

3.3.6.5 RADX Data Flow Analysis

The final static evaluation of the functional requirements specification is conducted using the Data Flow Analysis capability of REVS. This capability detects the incorrect use and assignment of information that flows through an R_NET, incomplete and ambiguous conditions for making branch decisions, and illegal references to SUBNET structures.

Information is identified as being incorrectly used if there is a condition which requires that it be INPUT to an ALPHA node, RECORDED by a VALIDATION_POINT node, PASSED by an OUTPUT_INTERFACE node, or used for making a decision at an OR node and there is no specification that establishes a value for the information prior to the node which uses it. Information is considered to have an established value if it has an INITIAL_VALUE, is PASSED by an INPUT_INTERFACE, is OUTPUT by an ALPHA, or is a member of an identified FILE or ENTITY.

The assignment of information that cannot be used is detected when an ALPHA OUTPUTS information and the information is reassigned before being used by a predecessor ALPHA, information that is part of a MESSAGE that PASSES an INPUT_INTERFACE is not referenced, or a MESSAGE is FORMed that cannot PASS an OUTPUT_INTERFACE.

The branch conditions of a CONSIDER OR node are tested to insure that they are unambiguous and exhaustive. When DATA with TYPE ENUMERATION is considered, a check is made to see that all possible values of the DATA specified in the RANGE altitude are contained in one and only one of the branch expressions that label the outward branches from the node. For the case that the considered element is an ENTITY_CLASS, the test is made that all ENTITY_TYPEs which COMPOSE the ENTITY_CLASS are represented exactly once in the branch expressions and there is only ENTITY_TYPE in the expressions that compose the class.

Partially rejoining AND constructs and partially rejoining OR constructs are prohibited by RSL within a single net structure. However, these illegal constructs can occur when a SUBNET containing a non-rejoining

logic construct is referenced within another net at a point where the reference is part of a rejoining construct. The Data Flow Analyzer accesses each reference to a SUBNET to identify the occurrence of this type of error.

3.3.7 Summary of Phase 3

This section has outlined a general procedure for completing the definition of the functional requirements for a DPS. The following steps were accomplished:

- The data transactions of each ALPHA were defined.
- The hierarchy transitions performed by each ALPHA, if any, were defined.
- The user has redefined ALPHAs into SUBNETs, has added ALPHAs, and has modified R_NET STRUCTUREs if the Phase 2 baseline was inadequate.
- The user has evolved a clear concept of the processing done within each ALPHA, as a starting point for development of BETAs.
- The definition of all necessary DATA and FILE attributes has been completed, to the extent possible without developing BETAs and GAMMAs.
- The completeness and consistency of the ASSM has been analyzed and verified with the aid of RADX capabilities.
- The REVS Data Flow Analyzer has verified the logical DATA flow and connectivity.

3.4 PHASE 4 - COMPLETION OF MANAGEMENT AND CONTROL INFORMATION

Traceability is a feature of the specification supporting its management, rather than one required for its technical quality per se. Therefore, the requirements for traceability are established in the management volume; however, they are addressed here in terms of requirements and techniques necessary for their entry in the ASSM.

3.4.1 Originating Requirements

The originating requirements for a software specification are most commonly contained in higher-level specifications. In general (depending on management decision) each identifiable software requirement in each higher-level specification will be called out as an ORIGINATING_REQUIREMENT in the ASSM. The DESCRIPTION attributed to an ORIGINATING_REQUIREMENT may be a literal excerpt of the source, or may be an interpretation, again at management discretion. Note that ORIGINATING_REQUIREMENTs in REVS do not trace to one another, however, an ORIGINATING_REQUIREMENT may be INCORPORATED IN a higher level requirement of which it is a part. The lowest level of requirements to which a DECISION or specification element traces should be entered as the ORIGINATING_REQUIREMENT. Appendix H illustrates the structuring of ORIGINATING_REQUIREMENT using the INCORPORATES and INCORPORATED IN relationships.

3.4.2 Sources

There may be sources of information which define specifics of the software requirements, but which are not specifications in themselves. For example, a table of constants, or a book defining a standard atmosphere model may be referenced in the source specifications or applied by the requirements engineer in developing the software specification. Such a SOURCE is also recorded in the ASSM, since it DOCUMENTs some feature of the required processing. Note that a SOURCE may be changed during development of either the specification or the software; when such a change occurs, its impact on the specification may be followed through use of the DOCUMENTs relationship.

A SOURCE may also DOCUMENT an ORIGINATING_REQUIREMENT, where its interpretation is significantly different. The collection of requirements in a specification document should be traceable to that SOURCE to support control of changes when the specification is modified; thus the SOURCE is the name of the document from which an ORIGINATING_REQUIREMENT is taken. When the SOURCE document is changed (as in a specification revision), the use of SOURCE can expedite tracking the influence of the change on the software requirements. (In a later section of this report, the concept of a PERFORMANCE_REQUIREMENT is developed. The specification serves as a SOURCE for the collection of PERFORMANCE_REQUIREMENTs in the same sense as for a collection of ORIGINATING_REQUIREMENTs.)

The RSL element VERSION is used to document a particular configuration of the system requirements definition. VERSION is used to trace data base elements which IMPLEMENTs a particular VERSION DOCUMENTED by a particular SOURCE. Appendix H illustrates the use of these elements and relationships to document contents of the ASSM.

3.4.3 Decisions

As was noted in 3.1.5, the process of developing the DPS requirements may expose system issues which cannot be resolved immediately in a formal manner, but which may have significant impact on the direction of further work. Each such issue is recorded in the ASSM as a DECISION which is TRACED FROM the ORIGINATING_REQUIREMENT(s) either directly or through other DECISIONs. As each issue is encountered it should be given a DECISION name, and entered in the ASSM with at least its key attributes: a statement of the PROBLEM, the recognized ALTERNATIVES, and the CHOICE among them which was made to allow work to proceed. When the CHOICE is reviewed by system engineering, its correctness can be confirmed, or the implications of any revision can be assessed through analysis of those elements which are TRACED FROM that DECISION. Paragraph 3.1.5 gives an example of the input needed to state one of the TLS issues.

3.4.4 Relating to Sources

The relationships TRACES TO and DOCUMENTS have been defined to link elements of requirements definitions to their sources. Each element of the ASSM should be TRACED FROM its ORIGINATING_REQUIREMENT(s) either directly or through DECISIONs which may be TRACED FROM the ORIGINATING_REQUIREMENT. In general, the ORIGINATING_REQUIREMENT may be entered into the ASSM before their traceability is developed (i.e., before any of the elements have been defined). As the R_NETs are built and as the other elements are entered, their links with any ORIGINATING_REQUIREMENT may be recorded through the TRACES TO relationship. Whenever a DECISION is encountered, it should be entered and TRACED FROM its source.

Chronologically, it is likely that the first entries made into the ASSM will be the ORIGINATING_REQUIREMENTs; some SOURCEs may also precede development of the R_NETs. As elements are developed, DECISIONs and additional SOURCEs will be recorded, and at least some TRACES TO and DOCUMENTs relationships will be provided. Before publication of the specification, management may require element audits to confirm some level of completeness of tracing; throughout development of both the specification and the resulting software, the linking should be monitored to track, in the specification, evolution of requirements. Note that it is good practice in requirements engineering to explore, through the data extractor, the impact of any projected modifications to requirements. Particularly at the terminal, it is convenient to query the ASSM to the effect: What if the DECISION (name) is modified? It would be accomplished LISTING ALL WITH an appropriate HIERARCHY that TRACED FROM that DECISION, then examining the attributes of each element that was so related. Judicious engineering could then focus on the requirements with the least impact when examining the set which might be altered to reflect a change in the software environment (e.g., the threat).

3.4.5 Informative Material

Phases 1 through 3 have discussed the development of the technical content of the ASSM. In addition to such mandatory material, there is a family of supportive information which might be recorded for any element. Characteristic of that family is the DESCRIPTION, an attribute which allows

an English-language text to be entered (other languages might be used except for restrictions due to character sets) to explain the intent of the element. Informative material is not constraining on the process designer, and is not, in fact, a requirement subject to test; it is merely supportive of communication between the requirements engineer and both his peers and the process designer. The need for any informative attribute and the auditing of its entry are options for project management, and are discussed in the management volume.

3.4.5.1 Description

A text string of arbitrary length may be entered to describe any element of the ASSM under the attribute DESCRIPTION. Even where the element carries a self-explanatory name, or where it has another attribute formally specifying it, a DESCRIPTION is usually valuable. For example, the IMAGE_ID is as simple a concept about an IMAGE as can be promoted. Nevertheless, it would be useful to describe it as being assigned from the HO_ID of the initiation message, and as being embodied also as the TARGET_ID in the ENTITY_CLASS: PULSE. The linkage among the elements is defined in the BETAs of appropriate ALPHAs, of course, and the DESCRIPTION is not binding. But following all of the references to that DATA item to find the meaningful assignments is tedious at best, where the DESCRIPTION is easily absorbed.

3.4.5.2 Synonym

A synonym may be declared and EQUATED TO any other element for clarity and convenience. Frequently, the SYNONYM will be a short name for a frequently referenced item, so that at least some of the references are simplified; occasionally, the naming conventions imposed by the language will prompt the use of a cryptic element name, so that an explanatory SYNONYM is desired.

3.4.5.3 Authorship

In the current REVS, there is an attribute ENTERED_BY which permits the author of information about an element to record his name and the date of entry. If required, that operation could be automated, so that the attribute would become a log of changes made, with the entry provided by the system whenever a value or relationship was altered.

3.4.5.4 Complementary Relationships

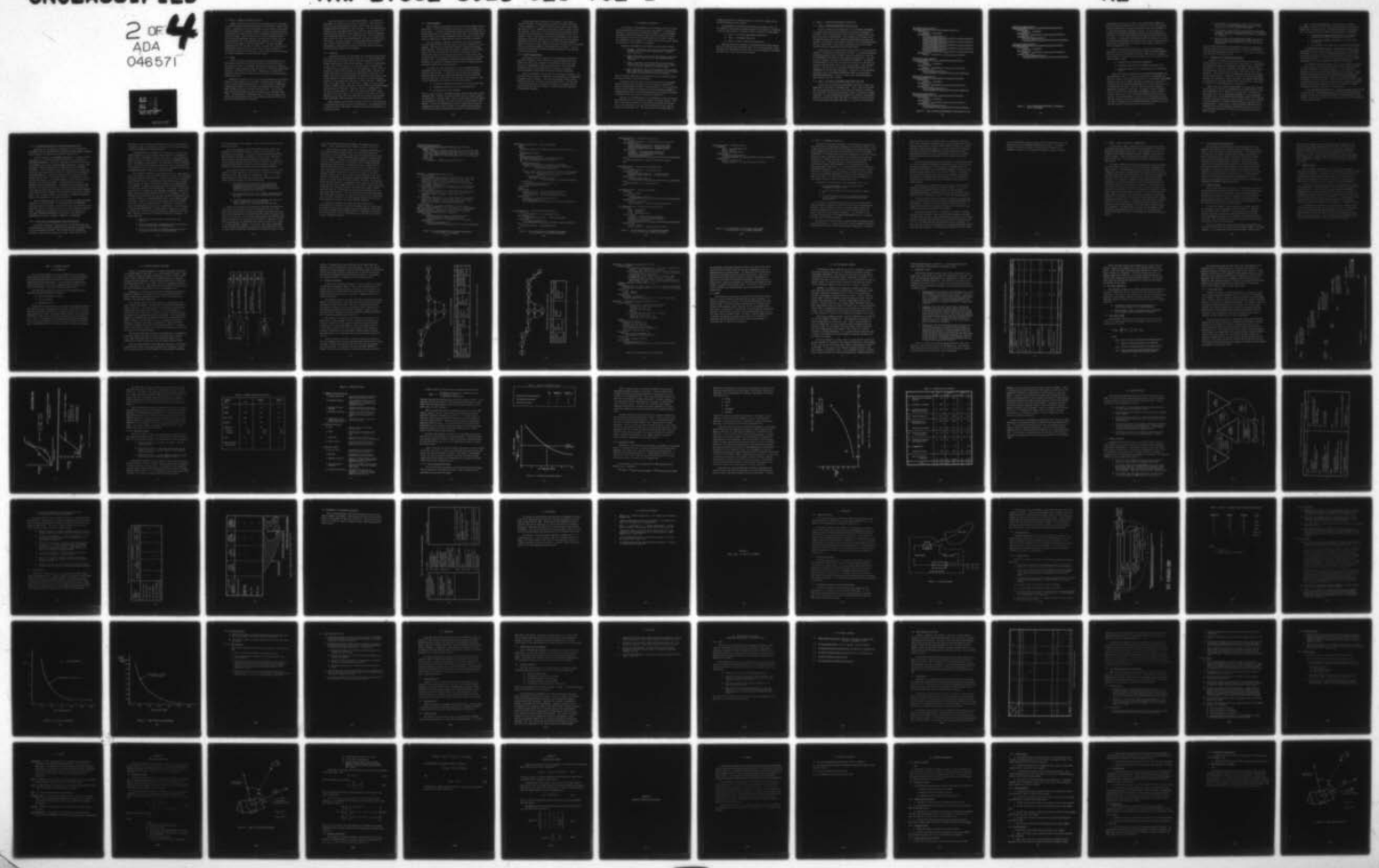
Each RSL relationship defines a connection between two elements; since it is transitive, it has a complement which simply reverses the subject and object. When information is extracted from the ASSM, both directions of the relationship are derived from a single link; thus, both the relationship and its complement are extracted, and redundant but absolutely consistent information is obtained. Since consistency is assured, redundancy is constructive in providing information about all relationships on an element under examination. Complementary relationships as a class may be suppressed in extraction through an APPEND PRIMARY.

3.4.5.5 Structural References

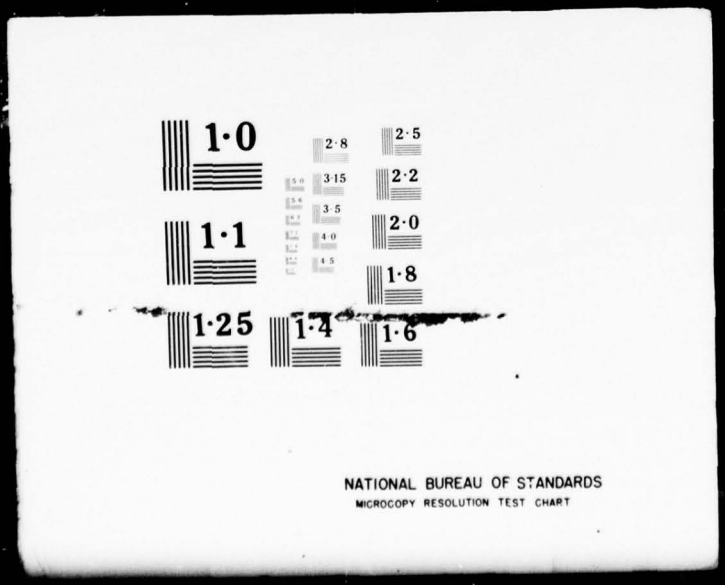
In the structure segment, an R_NET makes use of other elements. Implicitly, it has a relationship to each such element which appears on its STRUCTURE; that relationship is termed REFERS, and is implicit in use of the data extractor (RADX) of REVS. The relationship and its complement are visible to the user in the RADX output, but are entered automatically and implicitly through STRUCTURE declaration, not through the conventional explicit declarations.

AD-A046 571 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY. SREP FINAL REPOR--ETC(U)
AUG 77 M W ALFORD, P H BROWNE, I F BURNS DAS60-75-C-0022
UNCLASSIFIED TRW-27332-6921-026-VOL-1 NL

2 OF 4
ADA
046571



2 OF 4
ADA
046571



3.5 PHASE 5 - DYNAMIC FUNCTIONAL VALIDATION

Dynamic validation of the functional requirements entails development of functional models which define the outputs of processing in terms of the inputs. In essence, the only things known to a data processor specification are the contents of its interface messages; all other information should be defined within the specification in terms of mathematical operations on the input data stream. Such a definition is in effect a mathematical model of the processing operations required. Where the model reflects only the gross characteristics of the transformation required, it is said to be functional; such models in REVS are termed BETAs. The BETA and GAMMA representations of the DPS must be driven by a system level simulator which SREM assumes to exist, due to the prior need for such a simulator in deriving the originating specifications. The TLS example presumes existence of a System Environment and Threat Simulator (SETS).

3.5.1 Betas

A BETA is a PASCAL procedure which models the transformation of the DATA and FILEs INPUT TO an ALPHA into the DATA and FILEs OUTPUT FROM it. The purpose of a BETA is to implement the data flow required for functional simulation in order to determine the sufficiency of a functional specification. To that end, the fidelity of each BETA is dependent on the capabilities required of the functional simulator, and in particular on the data contents determined for SETS.

Generation of a BETA is more or less creative depending on the fidelity desired. At the simplest level, assignment statements may be employed in place of complex mathematical operations. In TSL, the ALPHA: UPDATE_STATE is modeled by assigning to STATE (a DATA item ASSOCIATED WITH ENTITY_TYPE: IMAGE_IN_TRACK) the value of the DATA returned by SETS. In an analytic model, the equivalent operation would be execution of a high-order Kalman filter. The simple model is appropriate since it follows the logical connectivity of the system data flow, while providing a means for SETS to activate the required branches of the R_NETs.

One of the ALPHAs of TLS is REDUN_DETERMINATION. It embodies the requirements for identifying two images as redundant if their state estimates are close enough (relative to their uncertainties) for them to be considered duplicate detections of the same object. The real processing to implement such a requirement would have to select at least a subset of all images then being tracked for state comparison, then determine redundancy by an appropriate algorithm. For a functional model, it is sufficient to define the USE of STATE (a multielement vector in reality) as BETA and its TYPE as REAL. Then two instances of IMAGE_IN_TRACK would be redundant if their values of STATE were equal. The corresponding SETS activity is implemented by having the DATA returned in the radar message selected to give the required probability of redundancy.

3.5.2 Local Data

In developing the BETA for each ALPHA, the requirements engineer must also solidify the attributes related to DATA flow. As already noted, some high-level DATA will be sufficient for functional modeling, and will be assigned the appropriate TYPE and USE attributes. Similarly, specification of USE: BOTH for DATA required in a functional model at the same level of fidelity as in an analytic model will be accomplished. Also, it will be necessary to identify DATA required for intercommunication of ALPHAs on a single R_NET during a single transaction (enablement) of that net. For example, the STATE of the IMAGE_IN_TRACK corresponding to the current return must be compared in REDUN_DETERMINATION with that of all other instances. Therefore, a DATA element may be defined as OUTPUT FROM ALPHA: UPDATE_STATE which is the CURRENT_STATE. That definition is needed since the current instance of IMAGE_IN_TRACK (with which a value of STATE is associated) changes during execution of the R_NET. Therefore, the DATA: CURRENT_STATE is declared, and given LOCALITY: LOCAL, TYPE: REAL, and USE: BETA. Since the value of the state vector is also an output for recording, and since a MESSAGE uses only LOCAL DATA, the element may be the same as the one which MAKES the MESSAGE: STATE_UPDATE which is also required from that R_NET.

RADX procedures (as discussed in Section 3.3.6) are of continuing use in analyzing the correctness of the DATA definitions. This applies not only to specific software requirements information but also to DATA defined for simulation purposes.

3.5.3 SETS Development

SETS is a generic name assigned to the driver for the software requirements models. Since SETS represents the totality of the environment to the data processor. Its design and development should normally be controlled by Systems Engineering or an independent V&V organization. SREM considers SETS to be an input to requirements development effort (although in reality SETS coding and debug may be accomplished by software requirements personnel).

From the software's viewpoint, SETS provides all stimuli necessary for each processing option and also accepts and properly executes all valid commands. SETS must also be designed to properly simulate the expected system timeline. Therefore, when an activity is commanded by the software models, SETS must be structured to 1) simulate the required action, 2) calculate how long the activity would have taken in the real system, and 3) make the results of the activity available to the software at the proper simulated time. Close coordination between the BETA developers and SETS designers will typically be necessary to ensure that the response timeline and timing interfaces are properly handled.

For Track Loop, time is controlled by SETS via commands from the requirements models and is administered by the REVS Simulation Executive using the Event Calendar. At the end of each activity, the Simulation Executive searches the Event Calendar and advances simulation time (RSL DATA item CLOCK_TIME) to the time of the next event to be processed. When an R_NET terminates, one of three actions may have occurred:

1) an EVENT was generated to enable an R_NET at some later time,

2) a MESSAGE was PASSED across an OUTPUT_INTERFACE, or

3) no future activity was scheduled.

Item (1) causes an entry in the EVENT calendar indicating the R_NETs future enablement time. An R_NET can therefore enable itself or another R_NET on an intermittent or periodic basis. Item (2) causes execution of SETS models and commands SETS to perform a specific function at a time defined in the MESSAGE. TLS SETS was implemented such that it establishes how long the commanded action should take to perform in the modeled subsystem and also at what time the results should be made available to the data processor.

Because BMD systems are generally asynchronous, R_NET timing is implicitly modeled. The data processing activity of scheduling actions of other subsystems (i.e., Radar) drives the system sensors much like an open loop servo (i.e., commands are transmitted, action is taken, results are returned). While SETS is executing the orders (and advancing simulation time) the Radar Scheduler is concurrently preparing the next set of commands. Since 1) REVS simulations consider requirements, not a specific architecture and process design, 2) it is assumed that the system imposed timelines can be met by the software, and 3) SETS has already advanced simulation time to the time of the next necessary software activity, simulation time is not directly advanced at the termination of an R_NET. It is controlled by the modeling of the data processor scheduling of the timelines of the other subsystems.

3.5.4 Simulator Applications

REVS provides the basic capability for the automatic generation of functional simulations. Also provided are interface software for integration with SETS, various methods of obtaining data during simulation execution and mechanisms to support post processing of the output data. The degree to which these capabilities are utilized on a program is directly dependent on the objectives of each particular application.

Track Loop represents only one level of a functional simulation. Its fidelity could be increased, if desired, and where necessary its orientation or emphasis could be changed. Track Loop was developed to meet certain objectives which may, or may not, exist on another effort. The key point is that each application is different and engineering judgement must be used to decide what level of simulator is needed to meet the objectives of the dynamic validation.

4.0 PERFORMANCE REQUIREMENTS

In the phases of SREM described in the following sections, the data processing specified by the RSL functional requirements is completed by definition of the necessary performance requirements. These performance requirements specify the conditions that must be satisfied during real-time processing in order for the system objectives to be met. They are also statements of the acceptability criteria that will be used for testing and validation of the software to be produced.

Performance requirements may be generally categorized as follows:

a) Accuracy - Requirements that specify the maximum acceptable software error in performing computations or in making logical decisions.

b) Range - Requirements that specify restrictions on the range (maximum value, minimum value) of data generated during processing.

c) Timing - Requirements that specify the time span allowed for processing, the times at which messages will be output, or the time spacing (frequency) of output messages.

c) Load - Requirements that specify the computational load that the software must be capable of handling, e.g., "the software must provide for concurrent tracking of 300 images".

Each of the above categories can be expressed in RSL to completely specify the desired software subsystem performance.

The approach described here for specifying performance requirements using RSL avoids most of the ambiguities and problems of untestability typical of specifications written in free-form English. It also avoids the issue of how to interpret the equations which are frequently included in software specifications. The techniques described here produce performance requirements which are explicitly testable and unambiguous.

The basis of the SREM approach is to unambiguously define the desired performance criteria as a precise TEST coded in the PASCAL language. Each TEST is constructed such that a single pass/fail assessment can be made. Data is first recorded at user specified VALIDATION_POINTs and then processed by the executable TEST. The TEST should be so designed that an

automatic evaluation can indicate success or failure with respect to the desired performance of the software.

The methodology for specifying performance requirements in RSL, in the context of the previously generated functional requirements, is described in the following sections. This part of SREM is performed in three phases:

- Phase 6 - Originating Requirement Decomposition
- Phase 7 - Performance Allocation
- Phase 8 - Analytic Feasibility Demonstration

These phases outline procedures for mapping DPSPR performance requirements onto the functional requirements nets, for allocating accuracy, range, timing and load requirements, and for verifying the completeness, consistency, and feasibility of the resulting specifications.

4.1 PHASE 6 - ORIGINATING_REQUIREMENT DECOMPOSITION

4.1.1 Step 1: ORIGINATING_REQUIREMENT Identification

The first activity in the generation of RSL PERFORMANCE_REQUIREMENTs is to isolate and identify each originating (performance) requirement in the DPSPR. Given a structured and well written system specification, this activity should be straightforward. Each statement of performance can be transcribed directly into an RSL ORIGINATING_REQUIREMENT and entered into the ASSM. If the DPSPR is sufficiently structured, each ORIGINATING_REQUIREMENT will address only a single area of performance and can be uniquely identified using the appropriate DPSPR paragraph number.

In many cases, this straightforward transcription will not be possible; therefore, an alternative scheme has been provided. If one DPSPR performance statement (paragraph) clearly addresses several different topics then the RSL INCORPORATES relationship is used to maintain traceability. This will, in effect, generate a hierarchy of ORIGINATING_REQUIREMENTs. The overall DPSPR requirement at the top of the hierarchy is declared in the ASSM, and references the applicable section of the source document. Each individual subtopic is then declared as a separate ORIGINATING_REQUIREMENT that is INCORPORATED IN the top level requirement. Traceability is thus maintained through the levels of ORIGINATING_REQUIREMENTs down to the applicable PERFORMANCE_REQUIREMENTs. Figure 4-1 shows example ORIGINATING_REQUIREMENTs generated from the TLS DPSPR.

4.1.2 Step 2: Preliminary PERFORMANCE_REQUIREMENT Definition

In this step, the ORIGINATING_REQUIREMENTs identified in Step 1 are mapped onto the functional requirements nets. That is, they are converted into the preliminary definition of TESTs that are formulated in terms of DATA collected at various VALIDATION_POINTs located on the R_NET and SUBNET STRUCTUREs. For each ORIGINATING_REQUIREMENT, the functional requirements defined by the R_NET and SUBNET STRUCTUREs are examined to determine those to which the ORIGINATING_REQUIREMENT applies. In some cases this mapping

```
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS,
     IMPLEMENTS:
          VERSION:  TLS_DPSPR,
     INCORPORATES:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_1_PERFORMANCE_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_2_PERFORMANCE_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_3_PERFORMANCE_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_4_PERFORMANCE_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_5_PERFORMANCE_REQUIREMENTS,
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT,

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_1_PERFORMANCE_REQUIREMENTS,
     IMPLEMENTS:
          VERSION:  TLS_DPSPR,
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT,
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS,

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_2_PERFORMANCE_REQUIREMENTS,
     IMPLEMENTS:
          VERSION:  TLS_DPSPR,
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT,
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS,

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_3_PERFORMANCE_REQUIREMENTS,
     IMPLEMENTS:
          VERSION:  TLS_DPSPR,
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT,
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS,
```

Figure 4-1 Sample ORIGINATING_REQUIREMENTs (Performance) for TLS

```
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_4_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_5_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.
```

Figure 4-1 Sample ORIGINATING REQUIREMENTs (Performance) for TLS (Continued)

(or decomposition) may be very simple: the ORIGINATING_REQUIREMENT may apply to a processing function that is confined to a single R_NET, and the TEST may be expressible in terms of existing DATA that can be RECORDED BY a single VALIDATION_POINT locatable on the net. For such cases, the resultant RSL PERFORMANCE_REQUIREMENT CONSTRAINs a single VALIDATION_PATH, consisting of a single VALIDATION_POINT that RECORDS existing DATA for use in the TEST. In other cases, the mapping may be quite complex, e.g., when the ORIGINATING_REQUIREMENT effects processing on several asynchronous nets, and requires the modification or addition of RSL elements in the ASSM in order to implement the TEST.

Step 2 is not easily configured into a step-by-step methodology, since many of the actions that are required in mapping ORIGINATING_REQUIREMENTs onto the nets are performed in parallel rather than in a stepwise, serial fashion. The activities to be performed in this step consist of the following general tasks.

- Decomposition of ORIGINATING_REQUIREMENTs
- Preliminary Definition of PERFORMANCE_REQUIREMENT TESTs
- Preliminary Definition of VALIDATION_POINTs and VALIDATION_PATHs

These activities are performed for each ORIGINATING_REQUIREMENT that was identified in Step 1. The activities are highly interdependent and should be viewed as being performed concurrently. The functional requirements defined by the R_NET and SUBNET STRUCTUREs are examined to identify the processing that the ORIGINATING_REQUIREMENT will (or might) constrain, and to identify DATA available on the nets that can be used to determine whether the requirement is satisfied. An attempt is made to formulate a pass/fail criterion for the ORIGINATING_REQUIREMENT in terms of DATA or FILEs on the nets that can be recorded at VALIDATION_POINTs during processing. The criterion sought is one that can be incorporated in an RSL TEST that explicitly specifies the conditions that recorded DATA and FILEs must meet. Depending on the ORIGINATING_REQUIREMENT, attempting to formulate the pass/fail criterion may lead to a decision that:

- The criterion is simply stated in terms of available DATA and FILEs and the ORIGINATING_REQUIREMENT should therefore decompose into a single PERFORMANCE_REQUIREMENT;

- The criterion is very difficult (or impossible) to formulate in terms of available DATA and decomposition of the ORIGINATING_REQUIREMENT into several PERFORMANCE_REQUIREMENTs is desirable (or necessary);

- The sensitivity of the ORIGINATING_REQUIREMENT constraint to DATA on the nets must be determined by further analysis in order to formulate the criterion and to define the required decomposition.

The activities described below, supplemented by studies or simulations as required, continue until the decomposition of the ORIGINATING_REQUIREMENT into PERFORMANCE_REQUIREMENTs has been defined. As previously stated, these activities are performed concurrently.

4.1.2.1 Decomposition of ORIGINATING_REQUIREMENTs

In general, the DPSPR will establish performance specifications at a higher level than that of the SREM functional specifications. Consequently, the methodology provides for performance requirements to be specified as constraints on a collection of functional requirements, thereby indirectly constraining each component function (R_NET, ALPHA, DATA, etc.) of the collection. This provision within the methodology supports design freedom for the process designer. It does not, however, preclude the requirements engineer from "decomposing" a DPSPR requirement into component requirements for allocation to individual elements within the collection of those R_NETs or ALPHAs to be constrained, should this be necessary.

When a high-level ORIGINATING_REQUIREMENT is analyzed during preliminary definition of its TEST, VALIDATION_POINTs and VALIDATION_PATHs, lower-level requirements that it implies will be identified. A decision must then be made as to how the ORIGINATING_REQUIREMENT is to be represented. For example, when a single ORIGINATING_REQUIREMENT applies to more than one functional requirement defined by R_NETs and SUBNETs that are not connected by EVENT enablements, the ORIGINATING_REQUIREMENT may constrain multiple VALIDATION_PATHs. Alternatively, a single ORIGINATING_REQUIREMENT may be decomposed into multiple PERFORMANCE_REQUIREMENTs which apply explicitly to the functional requirements defined by each non-connective R_NET and SUBNET STRUCTURE.

When an ORIGINATING_REQUIREMENT is decomposed into PERFORMANCE_REQUIREMENTs, these will each be TRACED FROM a DECISION that documents the decomposition. Decomposition of an ORIGINATING_REQUIREMENT necessitates that the following actions be taken to enter information in the ASSM.

- A DECISION that documents the decomposition via PROBLEM, ALTERNATIVES and CHOICE attributes will be declared.
- Each derived PERFORMANCE_REQUIREMENT will be TRACED FROM the DECISION.
- The DECISION will be TRACED FROM the ORIGINATING_REQUIREMENT.

A PERFORMANCE_REQUIREMENT imposes a quantitative constraint on the performance of certain processing paths. Decomposition of an ORIGINATING_REQUIREMENT into a set of derived PERFORMANCE_REQUIREMENTs requires that the constraint imposed by the original requirement be allocated to those that are derived from it. Clearly the combined constraints of the derived requirements must satisfy the constraint imposed by the original.

The allocation process may require only a simple mathematical analysis or it may require an in-depth trade study involving the construction and execution of functional or analytic simulators. If the allocation is reasonably simple, it can be performed immediately and the manner in which the resultant performance limits are obtained should be described in the DECISION that documents the decomposition of the original PERFORMANCE_REQUIREMENT into the derived ones. If the allocation requires that a large scale engineering trade study be performed, then the conditions and required outputs of the study are identified. The study is conducted in Phase 7 - Performance Allocation, described in the next section.

If the PERFORMANCE_REQUIREMENT TEST is completely defined except for the value of the limit to be imposed, then the TEST will be written except for insertion of the limit, and VALIDATION_POINTs and VALIDATION_PATHs will be defined. The limit for the TEST will be inserted in Phase 7 at the completion of the allocation study.

4.1.2.2 Preliminary Definition of Performance Requirement TESTs

Concurrent with the mapping and decomposition of the ORIGINATING_REQUIREMENTs and the identification of required allocation studies, attention must be focused on a preliminary definition of the RSL TESTs.

The TEST attribute of a PERFORMANCE_REQUIREMENT is a PASCAL procedure whose purpose is to explicitly define the criteria for satisfying the PERFORMANCE_REQUIREMENT. The TEST is formulated using DATA and FILES that are RECORDED by one or more VALIDATION_POINTs which appear as nodes on the STRUCTURE OF VALIDATION_PATHs that are CONSTRAINED BY the PERFORMANCE_REQUIREMENT. During a simulation, these DATA and FILES are recorded in an output data set and each record in the set is labeled with the appropriate VALIDATION_POINT name. During post-simulation TEST execution, the TEST accesses desired records in the output data set via the RETRIEVE, SELECT, and FOR EACH operators. Use of these operators in the writing of a TEST is described in the REVS Users Manual. The same DATA and FILEs may be RECORDED BY many VALIDATION_POINTs. To uniquely define accessing of DATA and FILEs from the output data set, all DATA and FILE names appearing in a TEST must be prefixed with the name of the VALIDATION_POINT which RECORDS those instances that are desired. The two names are separated by a decimal point. Thus, to refer to DATA (or FILE) A that is RECORDED BY VALIDATION_POINT V1, the identifier V1.A is used in the TEST.

Based on engineering analyses of the applicable requirements elements affected by a DPSPR requirement, it is may be possible to immediately define the precise TEST needed. In other cases the form of the TEST will be determined, but precise definition will occur in the Performance Allocation phase. Depending on the degree to which the TEST can be defined, the applicable PERFORMANCE_REQUIREMENT can be entered in the ASSM with a COMPLETENESS attribute of 'COMPLETE, INCOMPLETE, or CHANGEABLE'.

4.1.2.3 Definition of VALIDATION_POINTs and VALIDATION_PATHs

As the TEST formulation is being defined, the VALIDATION_POINTs and VALIDATION_PATHs with which information must be recorded to support the TEST can also be tentatively defined. The two operations are essentially concurrent, since they each depend on the other: there must be a TEST

definition in order to define the VALIDATION_POINTs and VALIDATION_PATHs to support it, but, at the same time, the TEST must be written in terms of information RECORDED BY the VALIDATION_POINTs.

A VALIDATION_POINT is analagous to a test point in a piece of electronic hardware; it is a port through which information is collected in order to assess performance of the unit under test. A single VALIDATION_POINT may be used to collect information for multiple PERFORMANCE_REQUIREMENTs; consequently, it may appear in the STRUCTUREs of several VALIDATION_PATHs. The DATA and FILEs declared as RECORDED by such a VALIDATION_POINT will therefore be the "logical OR" of those to be collected for each of the PERFORMANCE_REQUIREMENTs that use the VALIDATION_POINT.

The information RECORDED BY a VALIDATION_POINT may be simple DATA, an entire FILE, DATA or a FILE that MAKES a MESSAGE, or DATA or FILEs ASSOCIATED WITH ENTITY_CLASSes or ENTITY_TYPEs. For non-simple DATA (i.e., DATA which are CONTAINED IN a FILE, or are ASSOCIATED WITH an ENTITY_CLASS or ENTITY_TYPE), care must be taken in defining the instance of the repeated DATA to be RECORDED. In general, the instance desired will be selected earlier on the net that contains the VALIDATION_POINT. For example, in Track Loop, the PULSE which is relevant for the PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE is the one which was CREATED BY the ALPHA: PICK_COMMAND. Since no intervening activity modifies the selection, the DATA corresponding to that PULSE are available to the VALIDATION_POINT used for the ENERGY_PER_IMAGE requirement. However, the energy of that command, which is also needed for the TEST, is not so simply available. The energy of the PULSE is a part of WF_CHARACTERISTICS in the FILE: WAVEFORM_TABLE. The link between a PULSE and the WAVEFORM_TABLE is established by correlation of the type of transmission, identified in PULSE_TYPE (and also RADAR_TYPE) and in WF_NAME. There are several ways of obtaining the required information. An example method is:

1) Record either the PULSE_TYPE or the RADAR_TYPE from XMIT_R;

2) Add a VALIDATION_POINT: STARTING_POINT following the ALPHA: STARTER to RECORD the FILE: WAVEFORM_TABLE;

3) In the TEST, correlate the RECORDED information to obtain the energy of the PULSE from the WAVEFORM_TABLE.

The RSL associated with this example is shown as part of Figure 4-2 at the end of this section.

The correct placement of VALIDATION_POINTs requires careful interpretation of the DPSPR specification to ensure that the intent of the specification is preserved. If the exact intent of an ORIGINATING_REQUIREMENT contained in the DPSPR is ambiguous or unclear, then clarification should be sought from the systems engineering organization. If clarification cannot be obtained, then a DECISION that documents the PROBLEM, the various ALTERNATIVES and the CHOICE, should be entered in the ASSM. The DECISION is then TRACED from the ambiguous ORIGINATING_REQUIREMENT.

As an example, consider the specification statement in the TSL DPSPR (Appendix A) which requires that "The DPS shall allocate radar commands so that not more than (TBD) joules are commanded per image, ..." This apparently explicit requirement has at least three interpretations that require different placement of the VALIDATION_POINT.

1) The allocator shall assign track rates such that the accumulated sum of the energy for each image over the engagement (the product of the allocated pulse rate, the energy per pulse and the duration of the allocation) shall not exceed the specified limits;

2) The energy required by the radar commands transmitted across the interface to the radar shall not exceed the specified limit; or

3) The energy required by the radar commands acted upon by the radar, as detected by the DPS in the radar return messages, shall not exceed the specified limit.

Since not all of the pulses allocated per image may be scheduled and since some of the scheduled pulses may be preempted by the radar, the three interpretations lead to different performance requirement definitions. For the first interpretation, the VALIDATION_POINT should be located at the output of the allocator; for the second interpretation, the VALIDATION_POINT should be located at the input to the OUTPUT_INTERFACE: RADAR_OUT; for the third interpretation, the VALIDATION_POINT should be located at the output of the INPUT_INTERFACE: RADAR_IN. In the Track Loop example, development of the PERFORMANCE_REQUIREMENT was based on the second interpretation and is TRACED FROM a DECISION which delineated the development process, through

use of the DECISION attributes, the PROBLEM, the ALTERNATIVES and the CHOICE. The RSL associated with this example is shown in Figure 4-2.

Identification of a specific TEST for a PERFORMANCE_REQUIREMENT may require DATA that has not been defined in the functional requirement description of the DP subsystem. Also, it may be inconvenient to collect and correlate data from two VALIDATION_POINTs appearing on disjoint paths or net structures. Therefore, the requirements engineer may define DATA elements, ALPHA elements and, in some cases, R_NET or SUBNET STRUCTUREs with attributes ARTIFICIALITY: VALIDATION to convey the needed DATA to a VALIDATION_POINT where it may be recorded for a TEST. For example, in Track Loop, suppose that a performance constraint establishes a maximum delay time between the arrival of a track return message at the INPUT_INTERFACE RADAR_IN and the time at which the data contained in the return message is incorporated in the data which makes the command message that passes to the radar through the OUTPUT_INTERFACE RADAR_OUT. The connection between the two interfaces is asynchronous due to scheduling operations and use of the ENTITY_TYPE: IMAGE_IN_TRACK in R_NET: SKED_R. Since no one-to-one correlation exists between the return received and the command issued, a validation DATA item, RETURN_TIME, can be defined and ASSOCIATED WITH the ENTITY_TYPE: IMAGE_IN_TRACK. This DATA item would be OUTPUT FROM the UPDATE_STATE ALPHA, where it would be set equal to the current value of engagement time, and would be RECORDED BY a VALIDATION_POINT located on the the R_NET XMIT_R immediately preceding the OUTPUT_INTERFACE RADAR_OUT.

Information conveyed by an element type with ARTIFICIALITY: VALIDATION must be provided in the real-time process when it is being used to validate the process design against the system performance specification. In practice, element types so defined represent a counterpart to the hardware test point and, like their equivalent, may be retained in the fielded system by management directive.

```
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     DESCRIPTION:
     "DPSPR PARAGRAPH 3.2.4(B), RESOURCE CONTROL, STATES THAT
      ('THE DPS SHALL ALLOCATE RADAR COMMANDS SO THAT NOT MORE
      THAN(TBD) JOULES ARE COMMANDED PER IMAGE, NOR MORE THAN
      (TBD) KILOWATTS OR (TBD) PULSES/SECOND FOR ALL IMAGES IN
      TRACK.')",
     TRACES TO:
          DECISION:  RADAR_RESOURCE_CONTROL_R1



DECISION:  RADAR_RESOURCE_CONTROL_B1.
     ALTERNATIVES:
  "1. THE ALLOCATOR SHALL ASSIGN TRACK RATES SUCH THAT THE
      CUMULATIVE SUM OF THE ENERGY FOR EACH IMAGE OVER THE
      ENGAGEMENT (THE PRODUCT OF ALLOCATED PULSE RATE,
      ENERGY PER PULSE, AND DURATION OF ALLOCATION) SHALL
      NOT EXCEED (TBD) JOULES.
   2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED
      ACROSS THE INTERFACE TO  THE RADAR SHALL NOT EXCEED
      (TBD) JOULES.
   3. THE ENERGY REQUIRED BY THE RADAR COMMANDS ACTED UPON
      BY THE RADAR AS DETECTED BY THE DPS IN THE RETURN
      MESSAGES, SHALL NOT EXCEED (TBD) JOULES.",
     CHOICE:
  "2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED
      ACROSS THE INTERFACE TO THE RADAR SHALL NOT EXCEED
      (TBD) JOULES SHALL BE TESTED FOR COMPLIANCE AT THE
      INPUT TO THE OUTPUT INTERFACE: RADAR_OUT.",
     PROBLEM:
  "DPSPR PARAGRAPH 3.2.4(B), STATEMENT ('THE DPS SHALL
  ALLOCATE RADAR COMMANDS SO THAT NOT MORE THAN (TBD) JOULES
  ARE COMMANDED PER IMAGE,...') ALLOWS FOR THREE POSSIBLE
  INTERPRETATIONS IN DETERMINING THE POINT AT WHICH THE
  PERFORMANCE_REQUIREMENT TEST IS APPLIED.",
     TRACES TO:
          PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE
          VALIDATION_PATH:  RADAR_COMMAND_OUTPUT
          VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT,
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
```

Figure 4-2 RSL Representation of Performance Requirements at End of Phase 6 (TSL Example)

```
PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE.
     TEST:
     "CONST
       ENERGY_LIMIT=5.0;(* TEMPORARY REPLACEMENT FOR (TBD). *)
      VAR
       IMAGE_ENERGY: REAL;
      BEGIN
       ENERGY_PER_IMAGE:=TRUE;
       RETRIEVE FIRST RECORDING FOR STARTING_POINT;
       FOR EACH C2_IMAGE_HANDOVER RECORDING
       DO
         IMAGE_ENERGY:=0.0;
         FOR EACH RADAR_COMMAND_OUTPUT_POINT RECORDING
           SUCH THAT (RADAR_COMMAND_OUTPUT_POINT.TARGET_ID=
                      C2_IMAGE_HANDOVER.HR_ID)
         DO
           SELECT FIRST RECORD FROM STARTING_POINT.WAVEFORM_TABLE
             SUCH THAT (RADAR_COMMAND_OUTPUT_POINT.RADAR_TYPE=
                        STARTING_POINT.WF_NAME);
             IF RECORD_FOUND THEN
                IMAGE_ENERGY:=IMAGE_ENERGY+
                              STARTING_POINT.WF_CHARACTERISTICS;
         ENDFOREACH;
         IF (IMAGE_ENERGY>ENERGY_LIMIT) THEN
            ENERGY_PER_IMAGE:=FALSE;
       ENDFOREACH
      END;",
     CONSTRAINS:
          VALIDATION_PATH:  C2_HANDOVER_COMMAND_INPUT
          VALIDATION_PATH:  RADAR_COMMAND_OUTPUT
          VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES.
     TRACED FROM:
          DECISION:  RADAR_RESOURCE_CONTROL_P1
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.



VALIDATION_PATH:  C2_HANDOVER_COMMAND_INPUT.
     REFERS TO:
          VALIDATION_POINT:  C2_IMAGE_HANDOVER.
     CONSTRAINED BY:
          PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     PATH :
        VALIDATION_POINT:  C2_IMAGE_HANDOVER

     END.
```

Figure 4-2 RSL Representation of Performance Requirements at End of Phase 6 (TSL Example) (Continued)

```
VALIDATION_PATH:  RADAR_COMMAND_OUTPUT.
     REFERS TO:
          VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
     CONSTRAINED BY:
          PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE
          PERFORMANCE_REQUIREMENT:  PULSES_PER_SECOND
          PERFORMANCE_REQUIREMENT:  RADIATED_POWER.
     TRACED FROM:
          DECISION:  RADAR_RESOURCE_CONTROL_B1
          DECISION:  RADAR_RESOURCE_CONTROL_B2
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     PATH :
        VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT
     END.


VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES.
     REFERS TO:
          VALIDATION_POINT:  STARTING_POINT.
     CONSTRAINED BY:
          PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE
          PERFORMANCE_REQUIREMENT:  RADIATED_POWER.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     PATH :
        VALIDATION_POINT:  STARTING_POINT
     END.


VALIDATION_POINT:  C2_IMAGE_HANDOVER.
     RECORDS:
          DATA:  HO_ID.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     REFERRED BY:
          R_NET:  CC_RESPONSE
          VALIDATION_PATH:  C2_HANDOVER_COMMAND_INPUT.

VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
     RECORDS:
          DATA:  RADAR_TYPE
          DATA:  TARGET_ID
          DATA:  TRANSMIT_START.
     TRACED FROM:
          DECISION:  RADAR_RESOURCE_CONTROL_B1
          DECISION:  RADAR_RESOURCE_CONTROL_B2
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     REFERRED BY:
          R_NET:  XMIT_R
          VALIDATION_PATH:  RADAR_COMMAND_OUTPUT.
```

Figure 4-2 RSL Representation of Performance Requirements at End of Phase 6 (TSL Example) (Continued)

```
VALIDATION_POINT:  STARTING_POINT.
     RECORDS:
          DATA:  WF_CHARACTERISTICS
          DATA:  WF_NAME
          FILE:  WAVEFORM_TABLE.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     REFERRED BY:
          R_NET:  CC_RESPONSE
          VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES.
```

Figure 4-2 RSL Representation of Performance Requirements at End of Phase 6 (TSL Example) (Continued)

4.2 PHASE 7 - PERFORMANCE ALLOCATION

In Phase 6, performance requirements in the DPSPR were correlated with the software function requirements expressed by the R_NETs and SUBNETs. Based on the analyses performed, those DPSPR performance requirements which were not directly mappable onto the NETs were decomposed into a set of PERFORMANCE_REQUIREMENTs. The analyses upon which these decompositions were based varied in the level of detail considered, depending on the nature of the particular decomposition. Generally, however, they were on a functional rather than an analytic level. They were sufficient to determine the functional relationships between a DPSPR parameter to be constrained and the processing described by the R_NETs, i.e., they were sufficient for ORIGINATING_REQUIREMENT decomposition. In many cases, the analyses performed in Phase 6 were also sufficient to serve as the basis for allocation of performance limits to the PERFORMANCE_REQUIREMENTs derived from the decomposition. In some cases, however, the decomposition analyses must be supplemented with more in-depth studies in order to allocate these limits.

The purposes of the allocation studies performed in Phase 7 are:

1) To define performance limits to be incorporated in PERFORMANCE_REQUIREMENT TESTs,

2) To verify the decomposition analyses performed in Phase 6, and

3) If a TEST involves the use of a reference algorithm as a comparison standard for the software design, to define that algorithm.

The allocation studies determine the sensitivity of the originating DPSPR requirement to performance variations in the various functional areas affected. This is necessary in order to evaluate tradeoffs between the derived requirements and in order to select an allocation which is not overly restrictive in any particular dimension. It is desirable to use the least constraining set of PERFORMANCE_REQUIREMENTs that will guarantee satisfaction of the DPSPR requirement.

The allocation studies may cover a complete range of engineering analyses and techniques (i.e., they may consist of a manual mathematical analysis, they may require analysis support from functional simulations, or they may

require the construction of detailed models for an analytic simulation). Recall that a functional simulation has previously been generated from the functional specification using the SIMGEN function of REVS, and is readily available to support these analyses. If analytic models are constructed, these may later be used as GAMMA models for the Analytic Feasibility Demonstration (Phase 8), or may be incorporated in a TEST as a reference algorithm against which the performance of the software design may be assessed.

The allocation approach should be evaluated by a formal design review for invalid assumptions, inadvertent overconstraints, etc. This review is assisted by the capability of RADX to extract the traceability of the decomposition DECISIONs to the ORIGINATING_REQUIREMENTs, to the functional elements (R_NETs, ALPHAs, etc.), and to other DECISIONs. The adequacy of the projected performance models should also be reviewed to assure that all measures of performance have been accounted for. This should be a technical review conducted much like the preliminary design review for a software design.

The results of the allocation studies may lead to revision of the requirement decomposition that was performed in Phase 6. If so, then the decomposition DECISION that was declared, and the associated PERFORMANCE_REQUIREMENTs, VALIDATION_POINTs, and VALIDATION_PATHs that it TRACES TO, must be replaced or revised to reflect the new decomposition.

As each allocation study is completed, the results of the study are used to complete the TEST attributes of the PERFORMANCE_REQUIREMENTs that were analyzed. Study results are documented either directly in the decomposition DECISION or in a technical report which is declared in the ASSM to be a SOURCE which DOCUMENTs the DECISION.

After the allocation process is completed and the ASSM is updated, the RADX function of REVS is used to verify that the content of the ASSM is complete and consistent. When completeness and consistency are verified, the PPR is prepared using RADX to extract information from the ASSM and to list it in the format that has been selected by project management. At this point, the steps of the methodology for developing performance requirements have been completed. The ASSM data base has been built with liberal

use of the TRANSLATOR and RADX segments of REVS to ensure the accuracy and completeness of each PERFORMANCE_REQUIREMENT description. An analytic simulator can now be constructed to demonstrate the feasibility of a mathematical solution to the software requirements as stated in the PPR.

4.3 PHASE 8 - ANALYTIC FEASIBILITY DEMONSTRATION

At this point in SREM, a complete set of functional and performance requirements have been developed for the DPS and documented in the Process Performance Requirements specification. The PPR contains sufficient information for the process designer to design, implement and test the end product software. There is one major issue, however, that has not yet been resolved; that is, can an engineering solution be developed that will meet the PPR?

This phase of SREM provides for the demonstration of data processing subsystem analytic feasibility as a part of the overall requirements validation activity. The demonstration involves the design, implementation, integration, and test of a candidate design (including analytic algorithms or GAMMAs) which meets all the requirements described in the PPR except for those associated with timing. The objective of the feasibility demonstration is to 1) provide a "breadboard" design under the same logical requirements as the real-time process, 2) test this candidate design in a simulated environment, 3) evaluate the performance of the candidate process against the specified validation criteria, and thus 4) increase confidence in the feasibility of the PPR requirements by illustrating a single-point solution for the requirements specified. In areas where candidate GAMMAs cannot be developed, the corresponding requirement is potentially infeasible, thus identifying a problem area to be addressed at the system/subsystem requirements level. In addition to the establishment of analytic feasibility of the requirements, the candidate design can also be used directly as a design and validation tool by the process designers and the V&V organization. The GAMMA models, since they are guaranteed to effect the required transformations (input to output), can be augmented with execution-time models and used as initial models for the real-time process design.

4.3.1 Form of Feasibility Demonstration

The algorithms associated with the feasibility demonstration are developed and integrated into a candidate design which meets all requirements of the PPR (exclusive of timing requirements). The candidate design is in the form of an analytic non-real-time simulation, or emulation, which is exercised against a System Environment and Threat Simulation (SETS or similar subsystem models) simulating the threat, the system environment, and non-data-processing subsystems. The emulation should contain all of the validation points specified in the PPR; thus data is collected and performance evaluated under the same criteria as the real-time process. Further, the SREM SETS models can be constructed almost identically to the driver used to validate the real-time process and thus establish a common performance evaluation data base. Results of formal testing of the candidate design against a predetermined set of test scenarios can be documented along with any applicable unit tests that were conducted for individual algorithm validation.

4.3.2 GAMMA Development

Each candidate algorithm will generally be developed by an expert in the appropriate engineering or technology field. The design activity uses the descriptions of the RSL requirements to define what function each algorithm must accomplish, how well the function must be performed, and what information should be accepted as inputs and produced as outputs. Mathematical specifications of the algorithms are then implemented within the structure of the R_NETs as executable GAMMAs.

The PPR DATA items will usually be defined at a higher level than will be needed to build an excutable analytic simulation. In general, very few DATA with USE = GAMMA will, at this point, exist in the ASSM. The lower level data items must be defined and entered in the ASSM as either independent new elements or be INCLUDED in the appropriate higher level DATA. RADX can be used to verify the correctness of the new entries with respect to USE, TYPE, and LOCALITY.

After the algorithm (and data derivations) have been completed the mathematical relationships are implemented as executable GAMMAs in the PASCAL language. The executable descriptions look like PASCAL procedures with

omission of the procedure header and with augmenting statements for that part of the requirement representing access to data. All of the normal PASCAL programming features and facilities are available, except that all data that is not strictly local to a BETA or GAMMA must be explicitly declared in the ASSM. The REVS Simulation Generator processes these executable descriptions to produce standard PASCAL procedures for incorporation into the GAMMA level simulation. This is discussed in detail in the REVS Users Manual.

4.3.3 Benefits Derived

The explicitness, completeness, and testability of the software requirements are enhanced by the process of constructing a candidate solution. If additional assumptions or information about the system are required to accomplish the candidate design, these assumptions will be explicitly added to the requirements specification. This type of validation aids in determining the oversights in the requirements and system engineering.

An extremely important goal satisfied by the development of a feasibility demonstration is the establishment that the software requirements are actually testable. Since the candidate design applies the PERFORMANCE_REQUIREMENT TEST criteria in the same manner as the real-time process, the performance evaluation associated with the candidate design will necessarily isolate any requirements which cannot be tested. If a requirement is found to be untestable or unverifiable through the testing of the candidate process, the particular requirement will be changed to a testable criterion.

The development of an analytic point solution via a candidate design establishes the feasibility of the requirements criteria. This existence proof assures the process designer that there is at least one set of algorithms that satisfies the specified logical and computational requirements. The process designer is free to apply these candidate algorithms as appropriate to solution of the real-time process design problem.

PART II - MANAGEMENT APPROACH

5.0 INTRODUCTION

One of the major benefits in using the SREM tools and techniques to develop software requirements is that the process is inherently manageable. The technical approach consists of specific activities which have well defined beginning and ending points. Also the degree of automation provided by REVS allows a high level of management visibility which is ordinarily not attainable in the specification development process.

The three sections which follow discuss the more important aspects of managing software requirements engineering:

- Defining Measurable Milestones
- Cost and Schedule
- Management Control.

The emphasis in these sections is to describe the management considerations which are unique to the application of SREM. Just as Part I will not make good requirements engineers out of technicians, Part II will not make good managers out of clerks. Even good managers, however, can be relatively ineffective if they do not have the means to establish visibility and control. Part II is intended to give the experienced manager an understanding of how the tools and techniques of SREM can be used to establish and maintain a sound management plan for the specification of software requirements.

6.0 DEFINING MEASURABLE MILESTONES

The key to successful management of software requirements engineering is the establishment of meaningful, clear, and measurable milestones. There is a tendency to treat requirements generation as a level of effort problem since the job of milestone definition is somewhat nebulous and imprecise using conventional approaches. Establishing milestones such as "Processing Performance Specification - First Draft", Processing Performance Specification - Second Draft", etc., may provide clarity and superficial measurability but does not establish a meaningful or effective management tool.

Management of an activity using the Software Requirements Engineering Methodology (SREM) can be extremely effective because the discipline inherent in SREM permits segmenting the effort into activities which have meaningful and measurable terminations.

As an example, consider the SREM procedures for generation of functional software requirements. An overview of the applicable SREM phases is shown in Figure 6.1. The initial activity is a preliminary evaluation and organization of the source specifications. This is accomplished by an initial definition of the R_NETs. Once this activity is complete, parallel development of the segmented requirements can proceed. As the requirements segments are developed, they are entered into the REVS data base (ASSM) for static evaluation. When the ASSM entry for all software requirements is complete, a functional simulation is generated and a dynamic evaluation performed to complete the functional requirements validation. At various points in this process, problems in the source specifications are identified and fed back to the systems engineer.

Each of these five phases ends with the development of an intermediate product and thereby constitutes an initial set of high level milestones. As was shown earlier, the status of these intermediate products can be automatically assessed in many areas using the support features of REVS.

Given that the completion of each SREM phase represents a significant milestone within itself, a much finer decomposition is necessary to effectively manage and control the sub-elements of the requirements development

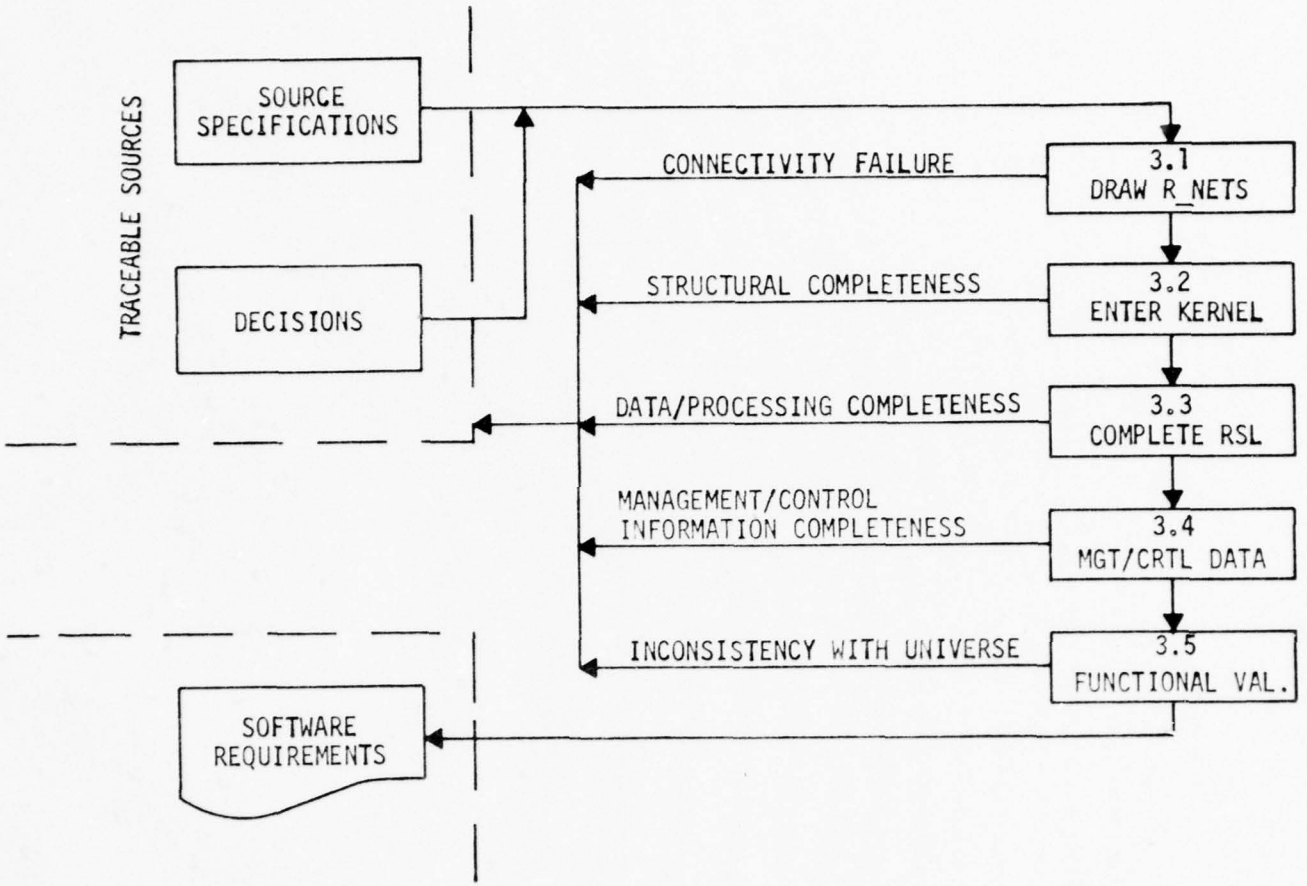


Figure 6-1 Overview of SREM Activities
(Development and Validation of Functional Requirements)

process. To accomplish this, each SREM phase can be broken down into a definitive set of milestones via a two-step process. First, the specific technical activities necessary to develop the desired product are identified. Then the key review and control points peculiar to the specific program are defined and integrated with the technical milestones. An example of this process for Phase I is described in the following sections.

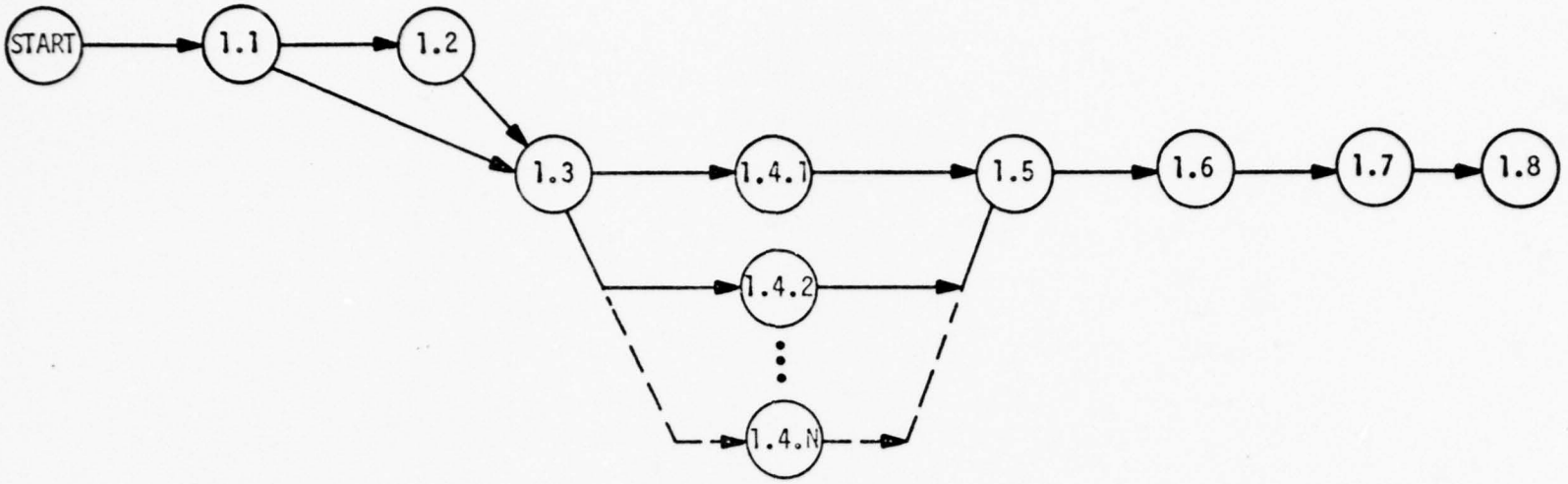
6.1 TECHNICAL MILESTONES

A sample set of technical milestones is shown in Figure 6-2 for Phase I, Definition of Subsystem Elements. Each of the milestones was derived directly from the discussion of Phase I activities given in Section 3.1. Eight milestones have been identified in this sample; for different applications these could be added to, or modified, to emphasize specific programmatic objectives.

The beginning point for the overall software requirements development process is the baselining of the system level source specification by the Configuration Control Board (CCB). Once this is accomplished, technical activities are focused on clearly specifying the direct interactions of the software subsystem with its external environment. Three milestones were specified for completion of the RSL INTERFACE and MESSAGE definition and the necessary supporting data hierarchy.

One milestone is provided for the completion of each R_NET to the ALPHA level. Since the number of R_NETs is driven by the necessity to accept or produce data at the subsystem interfaces (which were defined at milestone 1.1), the required number of nets can be determined early in the planning process. The final four milestones in the example are directed to completion of the requirements information/data structures and documentation of all engineering decisions that were made during the activity.

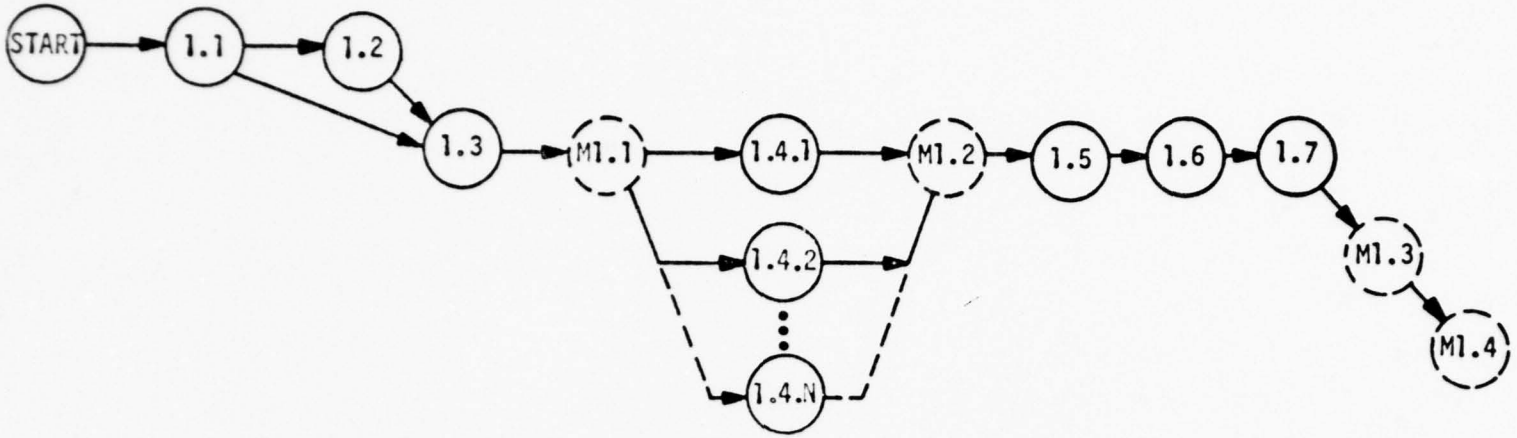
Figure 6-2 does not represent a series of technical activities in which the predecessor must be fully completed before the next effort can be initiated. In most cases overlap of the activities would be scheduled as soon as possible. For example, after each individual INTERFACE is defined, then the work can proceed on formalizing the precise content of



TECHNICAL MILESTONE DESCRIPTIONS

START	1.1	1.2	1.3	1.4.1---1.4.N	1.5	1.6	1.7	1.8
SOURCE SPECIFICATIONS BASELINED BY SYSTEMS ENGINEERING CCB	INTERFACE DEFINITIONS COMPLETED	MESSAGES DEFINED	INTERFACE DATA HIERARCHY SPECIFIED	R_NET #1 R_NET #2, ETC., COMPLETE TO ALPHA LEVEL	ENTITIES AND ASSOCIATED DATA DEFINED	FILE DEFINI-TIONS COMPLETED	ENGINEERING DECISIONS COMPLETED/ DOCUMENTED	R_NETS COMPLETED (REFINED AS NECESSARY)

Figure 6-2 Sample (Technical) Activity Network for SREM Phase 1



MANAGEMENT MILESTONES (ADDED TO FIGURE 6-2)

M1.1	M1.2	M1.3	M1.4
INTERFACE DEFINITION REVIEW WITH SYSTEMS ENGINEERING	DATA BASE APPROVAL AT ALPHA LEVEL	DECISIONS APPROVED BY SRE CCB	SOFTWARE REQUIREMENT INTERNALLY BASELINED BY SRE CCB

Figure 6.3 Complete Sample Activity Network for SREM Phase 1

```
DECISION:  RADAR_SCHEDULER_PRIORITIZATION.
     ALTERNATIVES:
               "1. SCHEDULE PULSE_BY_PULSE.  THIS WOULD SIMPLIFY THE
                 NETS BUT WOULD ORVIATE OPTIMIZATION.
               2. OPTIMIZE OVER THE ENTIRE FRAME.  TAKING THE FRAME
                 AS A WHOLE GIVES BEST RESULTS BUT REQUIRES WEIGHTING
                 FACTORS FOR PULSE ENSEMBLES.
               3. PRIORITIZE PULSES SUCH THAT ANY PULSE OF HIGH
                 PRIORITY BEATS ALL PULSES OF LOWER.  THIS IS
                 SUBOPTIMAL, BUT REALIZABLE BOTH IN THE SPEC AND IN
                 THE SOFTWARE DESIGN.  NO A PRIORI WEIGHTS NEEDED.".
     CHOICE: "OPTION 3. PRIORITIZED PULSES".
     PROBLEM:
         "OPTIMIZATION OF RADAR USAGE REQUIRES A FINITE RADAR FRAME.
         THIS IMPLIES A PRIORITIZATION SCHEME FOR INTENDED ORDERS.".
     TRACES TO:
          R_NET:  SKED_R
          R_NET:  XMIT_R.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.

DECISION:  SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK.
     ALTERNATIVES:
               "1. SYNCHRONOUS TRACKING (OR RESPONSIVE)
                  REQUIRES THE LAST RADAR RETURN ON
                  AN IMAGE BE USED TO
                  PRODUCE THE NEXT RADAR
                  ORDER.
               2. ASYNCHRONOUS TRACKING %OR AUTOGENIC<
                  ALLOWS A TRACK PULSE
                  TO BE SENT USING WHAT EVER
                  STATE IS IN THE DATA BASE.".
     CHOICE:
       "ASYNCHRONOUS TRACKING IS
          SELECTED TO MAXIMIZE THE
          ALLOWED DP TIME RESPONSE
          FOR PROCESSING RADAR RETURNS.
          THIS DOES NOT PROHIBIT A RESPONSIVE
          TRACKING IMPLEMENTATION.".
     PROBLEM:
        "TRACKING CAN BE EXPRESSED AS
             SYNCHRONOUS OR ASYNCHRONOUS.".
     TRACES TO:
          ALPHA:  PICK_CANDIDATES.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIRENENTS.
```

Figure 6-4 Sample Decisions from Track Loop

not the software requirements adequately reflect the intent of the source specifications and to what extent the software requirements provide an appropriate base for process design. The former issue can be resolved by analyzing the DECISIONs against the source specification. These can be categorized into those with major effect on software requirements and those with minor effect on the software requirements. Baselining may be deferred until certain major problems are resolved and the number of minor problems is reduced to an acceptable level. The visible act of deferring a major milestone can bring considerable pressure to work problems expeditiously.

6.3 SUMMARY

The preceding was an example of how the SREM activities described in Section 3.1 can be related to measurable and meaningful milestones. An identical approach can be used for the activities described for the other phases of SREM. The activity networks presented are not intended to be a universal SREM management plan in which one can fill in dates and names and be done. Every program is different as is each engineering organization. Therefore, there cannot be a universal milestone/management plan any more than there can be a universal R-NET. The point is that the SREM's approach to software requirements development and validation is structured and formalized such that meaningful milestones can be readily defined -- milestones that are specific and measurable.

7.0 COST AND SCHEDULE PLANNING

The formalized steps of SREM and the specific intermediate milestones which result provide a sound basis for scheduling and bugeting software requirements development. This section provides models and guidelines for estimating the cost of developing software requirements using SREM. The experience data base is as yet too small to fully calibrate and validate the models; however, cost data has been collected on three SREM applications, and the correlation between that data and the cost models is provided.

SREM/REVS is a composite of a methodology, language and supporting tools that allow identification of key milestones in a development program and which generates an output whose size is realistically quantifiable. Consider a requirements paragraph written in English. It can be measured by the number of words or sentences, but these parameters do not scope the content in any meaningful way. Conversely, requirements written in RSL consist of specific elements such as R_NETs, ALPHAs, INTERFACEs, DATA, etc. While these elements may vary slightly in length, size or implied complexity, they do offer a reasonably consistent set of parameters for specification sizing. Given the magnitude of the product to be developed, then cost can be related directly to the work to be accomplished to generate that product.

SREP cost and schedule estimation involves establishing cost parameters for each RSL ELEMENT or ELEMENT_TYPE. These parameters are developed in much the same manner as cost-per-unit code. After a cost-per-element or cost-per-element-type is derived (based on previous experience), then the accuracy of SREP cost and schedule estimates becomes a direct function of the initial understanding of the problem. This situation is somewhat analogous to software development, i.e., accurate cost-per-instruction data is useful to the extent that initial estimates of how many statements will be required is accurate.

Using this approach to costing, there is a necessity for an early and accurate estimate of the number of RSL elements required for a given program. In fact, this was one of the underlying issues in the definition of Phase I of the SREM. In this phase a quick response primitive construct of the major elements is completed. Phase I can be conducted on paper, or entered into a working data base with machine-processable RSL, allowing maximum use

of REVS automated graphics capabilities. Either method provides for rapid collection of information for end product sizing.

7.1 EXPERIENCE TO DATE

SREP has been applied to several small efforts (less than 1-2 man-months) and to two medium scale projects (1-2 man-years in duration). The small applications were demonstration type problems that focused on specific subsets of SREM/REVS capability. One, referred to as Project B, was developed to the extent that its resource utilization data is meaningful. Three data points are therefore available to assist preliminary formulation of SREM cost models:

- Track Loop was the first medium scale SREM application. Track Loop represents a well-defined system for which a specification of reasonably high quality was available. The system's mission is a subset of a full ballistic missile point defense role: included are the tracking, resource management and scheduling functions. A software implementation of Track Loop requirements is estimated to be 25-40K of program source statements.

- Project A, the second medium level effort, generated the requirements for a tactical weapons system. The weapons system was distributed in nature and featured various types of air defense subsystems interacting with multiple airborne attackers and offensive missiles. This effort was conducted early in the project's Concept Definition phase; consequently, requirements were modified many times due to changes at the system level. It was estimated that when the Project A software subsystem is implemented, it will contain approximately 200,000 source statements.

- Project B was the only small demonstration problem of sufficient scope to include for cost and schedule information. The system configuration consisted of a data processor and radar whose functions were to collect telemetry data from airborne weather balloons. System operation was such that the data processor controlled an uplink from the radar to the weather balloon for beacon tracking purposes and a downlink for the monitoring of temperature, wind, pressure, etc., at various altitudes. No estimates have been made on the size of the software subsystem for Project B.

Table 7.1 itemizes the resources expended on each of these three projects according to the phases of the SRE Methodology. Note that the small demonstration effort (Project B) is listed in man-hours as opposed to the other two projects which are listed in man-weeks.

Table 7.1 Resources Expended on Completed SREM Applications

METHODOLOGY PHASE	TLS (MW)	PROJECT A (MW)	PROJECT B (MH)
I. Definition of Subsystem Elements	2	13	3
II. Evaluation of Kernel	5	4	8
III. Completion of Functional Description	10	15	11
IV. - Documentation	1	2	8
- Informative Material	4	12	0
V. - Development of Functional Models	12	22	8
- Requirements Testing	6	25	24
MISC - Simulation Driver	8	15	24
	48 MWs	108 MWs	86 MHs

Each of the three efforts were performed using a uniform staffing profile. Project B was conducted by a single engineer during a 3-week period. Of the two larger efforts, both were conducted at a steady state level of either 2 or 3 engineers. At the start of both efforts, one engineer generated the definition of subsystem elements and derived the initial R_NETs connecting the input and output interfaces. The staffing levels were then increased and maintained at a constant rate throughout the duration of each effort.

7.2 COST MODEL FORMULATION

Two approaches to formalizing a SREM cost model have been formulated. The first is theoretical and will require significant data for accurate calibration. The second is less complex and can serve as a interim approach. Both use, as a key element, the basic resources necessary to generate a measurable quantity of output and both were developed to meet the following objectives.

- End product size estimates must be generated as an initial part of Phase I of the SREP methodology.
- Overall program resource requirements must be decomposed and allocated to defined intermediate milestones.

7.2.1 Cost Model 1 (CM1)

Because Software Requirements Engineering is basically a creative process, it closely parallels software development; this implies a cost relationship of the form

$$Cost_{SRE} = \sum_{j=1}^{P} (BCU_j \cdot \prod_{i=1}^{3} IC_{ij}) + C_{SR}$$

where

BCU_j = Basic cost unit (man-months) per methodology phase

P = Number of methodology phases to be conducted

IC_{ij} = The set of intangible factors that modify the BCU

C_{SR} = Cost of developing peripheral/supporting items necessary for completion of the SRE effort.

This relationship shows that each phase of the SREP methodology has associated with it a certain basic cost. The basic cost reflects the resources necessary to complete that phase under average conditions. Since average conditions tend to be atypical on most projects, the basic cost of each phase can be adjusted (either higher or lower) by the values of three parameters that reflect a set of intangible (or off-normal) constraints. Supporting items developed external to the mainline requirements engineering function (e.g., tactical algorithms) are added directly to the adjusted basic cost to determine overall resource requirements.

7.2.1.1 Basic Cost Units

The Basic Cost Unit for software requirements is calculated in terms of the resource expenditure related directly to the size of an intermediate or end product. Figure 7-1 presents a chart of the phases of the SRE methodology. The Basic Cost Unit associated with each phase is related to specific measures of the quantity of output from that phase. For example, in Phase III, the number of ALPHAs and DATA items drive the Basic Cost for that phase. Although many different functions are performed in this phase, and other specifications items generated, these are the two controlling parameters. Therefore, given an estimate of the number of ALPHAs and DATA quantity required, the Basic Cost of Phase III can be determined.

The upper section of Figure 7-2 defines the postulated relationship between the BCU and the number of RSL elements to be specified. As shown, the curve is anticipated to increase as a damped exponential prior to Point N; the basic shape of the curve will then change to a fast-rising exponential form. This indicates that, up to a certain point, the addition of one element to the data base will cause an increase in resource expenditures at less than a linear rate. Once Point N has been reached, any additional element added to the data base will cost more than one unit of resources. The value R_o accounts for startup time and indicates that even for limited productivity the resources expended in learning and understanding the problem are the dominant factor.

PHASE I. DEFINITION OF SUBSYSTEM ELEMENTS

Primary Activities

Preliminary Definition of interfaces, processing flow, hierarchies and data. Link input-output interfaces.

PHASE II. KERNEL EVALUATION

Establish data naming, enter all elements in data base, perform manual and automated analysis (RADX).

PHASE III. COMPLETION OF FUNCTIONAL DEFINITION

Complete specification of data transactions and hierarchies, ALPHAs, SUBNETs and R_NETs, and Data/File Definitions. Perform RADX static analysis.

Contributors to Basic Cost Unit

R_NETs
ALPHAs
MESSAGES

R_NETs
ALPHAs
MESSAGES

PHASE IV. COMPLETION OF MANAGEMENT AND CONTROL INFORMATION

Connect traces to originating requirements and references. Complete linkage of decisions. Complete descriptions, authorship, etc.

ALPHAs
DATA

All Applicable Elements

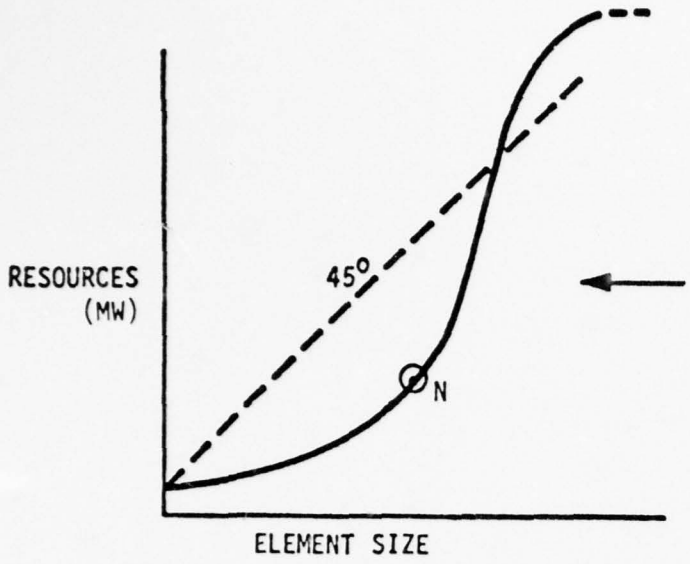
PHASE V. DYNAMIC FUNCTIONAL VALIDATION

Define Beta procedures. Add required data specification and flow. Develop functional SETS. Execute functional simulation.

BETAs
SETS*

*Supporting/Peripheral

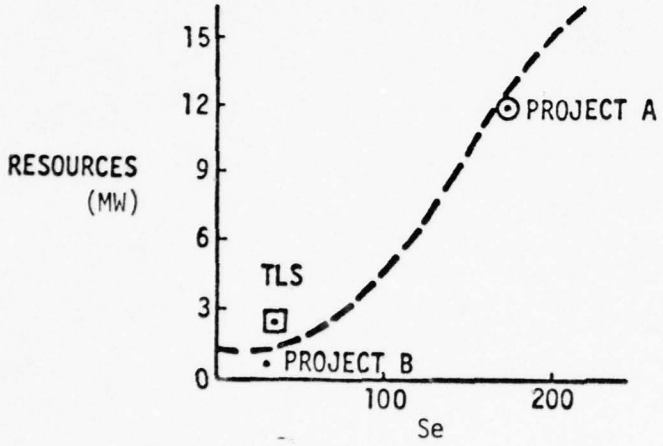
Figure 7-1 Primary RSL Elements Affecting Basic Cost (Functional Requirements)



EXPECTED GENERAL FORM

$$Ro + \begin{cases} e^{Se/K_1} & Se < N \\ e^{K_2 S_e} & Se \geq N \end{cases}$$

Se = f(# OF APPLICABLE RSL ELEMENTS)
[SEE FIGURE 7-1]



Se = MESSAGES + R_NETS + ALPHAS/2

Se (TLS) = 35
Se (PROJECT A) = 176
Se (PROJECT B) = 16

METHODOLOGY PHASE I

Figure 7-2 Basic Cost Unit Relationships

The second half of Figure 7-2 shows the correlation of the three available data points for SREM Phase I application on Projects A and B and for TLS. For Phase I, the independent parameter (Se) was assumed to be a linear function of the number of MESSAGEs, R_NETs, and ALPHAs. The count of the first two elements have been combined with the number of ALPHAs divided by two to indicate that approximately 50 percent of the ALPHAs are defined in Phase I, and the remaining 50 percent completed in Phase III.

Table 7.2 presents the number of requirement elements for each of the three projects. From this information, the exact number of elements generated in each SREM phase can be determined. The three data points are plotted in Figure 7-2 in man-weeks versus the effective number of elements generated. As shown, these results potentially conform to the expected shape of the relationship curve. By selecting the appropriate combination of cost elements, this exercise could be completed for each phase. This would, however, be somewhat academic due to the limited number of data points currently available.

7.2.1.2 Intangible Factors

There are a number of intangible factors that will affect the cost of a creative development process. For the purposes of SREM Cost Model One, all of the intangible factors have been grouped into three categories:

1) System Specification(s) Quality - How "good" is the SRE input information?

2) DP Subsystem Complexity - Will the complexity of the end product be "normal", relatively simple or complicated and highly interactive?

3) Development Factors - Is there something about the way the job must be performed that will impact the cost?

Given these categories the next question is how can they be quantified or rated for a specific program? Since the rating will, of necessity, be intuitive, there is no easy answer. To reduce some of the ambiguity, a second level breakdown of the contributing factors in each main category has been defined (Table 7.3). The subcategories indicate the general class of factors that should be considered under each major grouping.

Table 7.2 Elements Generated Per Project

ELEMENT_ TYPE	TLS	PROJECT A	PROJECT B
R_NETS	7	55	2
ALPHAs	30	148	13
DATA ITEMS	100	333	68
MESSAGES	13	47	7
INTERFACES	6	32	4
• Input	(3)	(19)	(2)
• Output	(3)	(13)	(2)
FILES	10	33	3
ENTITY_CLASSES & ENTITY_TYPES	8	7	5

Table 7.3 Intangible Factors

- System Specification(s) Quality
  - Interface Definitions - Have the DP interfaces with other subsystems been adequately specified?
  - Functional Integrity - Has the required processing been logically connected or specified as an unrelated series of activities to be (somehow) performed?
  - Performance, Timing, Accuracy - Are there quantitative measures for 'how well' the software must perform?
  - Environment Description - Quality of the specification of the environment external to the overall system (of which the software is a part)
  - General Organization, Consistency, Completeness, Level of Detail, etc. - Aggregated assessment of all factors other than the above.
- DP Subsystem Complexity
  - Type - Batch vs distributed, off-line vs real-time, etc.
  - Threat - Effects of the system mission, threat size, complexity, and imposed timelines
  - Interactions - Assessment of the logical flow, connectivity, number of paths and branches, etc.
  - State-of-the-Art - Is this a new type of system with unknown control processes or does all required technology already exist?
- Development Factors
  - Personnel Talent - Assessment of the basic capabilities of the contributing individuals
  - Motivation - Any unusual factors that add to, or subtract from, personnel motivation
  - Management Procedures - Effects of government imposed standards, documentation reporting schemes, configuration management, etc.
  - Artificial Schedule Constraints - External requirements to proceed faster than a realistic timeline or the delay of key inputs
  - Key Element Availability - Assessment of the access to, or availability of, computers, systems engineers, input data, skill centers, technology information, etc.

A scale of zero to 10 was selected to quantify each major group:

Scale -----
- 10 highest positive value, subtracts from cost
- 5 average, no effect
- 0 worst case, adds to cost

The Basic Cost Unit was defined as the cost of an 'average' effort. If all intangible factors were rated at five, the BCU would not change. Values higher than five would decrease the net cost and values lower than five would cause an overall increase.

Table 7.4 shows the composite intangible rating for the three SREM projects conducted to date. Each rating was determined after program completion following the guidelines in Table 7.3. As shown, Project A, which was conducted during Concept Definition, was rated relatively low. The system specification documents were not firm and, with each update, additional effort was necessary to revise the evolving DP requirements. Developmental factors were also rated low because of a large geographical separation between the various participating agencies plus a limited access to the necessary computer resources.

For these three projects the values of the intangible ratings were assessed to be constant across all methodology phases. Although the basic cost equation allows for variable ratings, the pertinent chararacteristics of these SREM applications did not vary sufficiently per phase to impact the assigned values.

Figure 7-3 depicts the predicted impact relationship of the IC ratings on the Basic Cost Units. As shown, the predicted relationship follows the theory that 'the bad outweights the good'. For example, if Development Factors were very constraining (rated at 1), then the value of a good system specification could be more than offset by a possible 100 percent increase due to poor Development Factors.

7.2.1.3 Cost of Supporting Elements

There are two major contributors to overall cost that can be classified as supporting or peripheral activities: SETS (simulation driver) development and algorithm development.

Table 7.4 Rating of Intangible Factors

	TLS	PROJECT A	PROJECT B
System Specification Quality	7	2	7
DP Subsystem Complexity	7	5	8
Development Factors	6	3	9

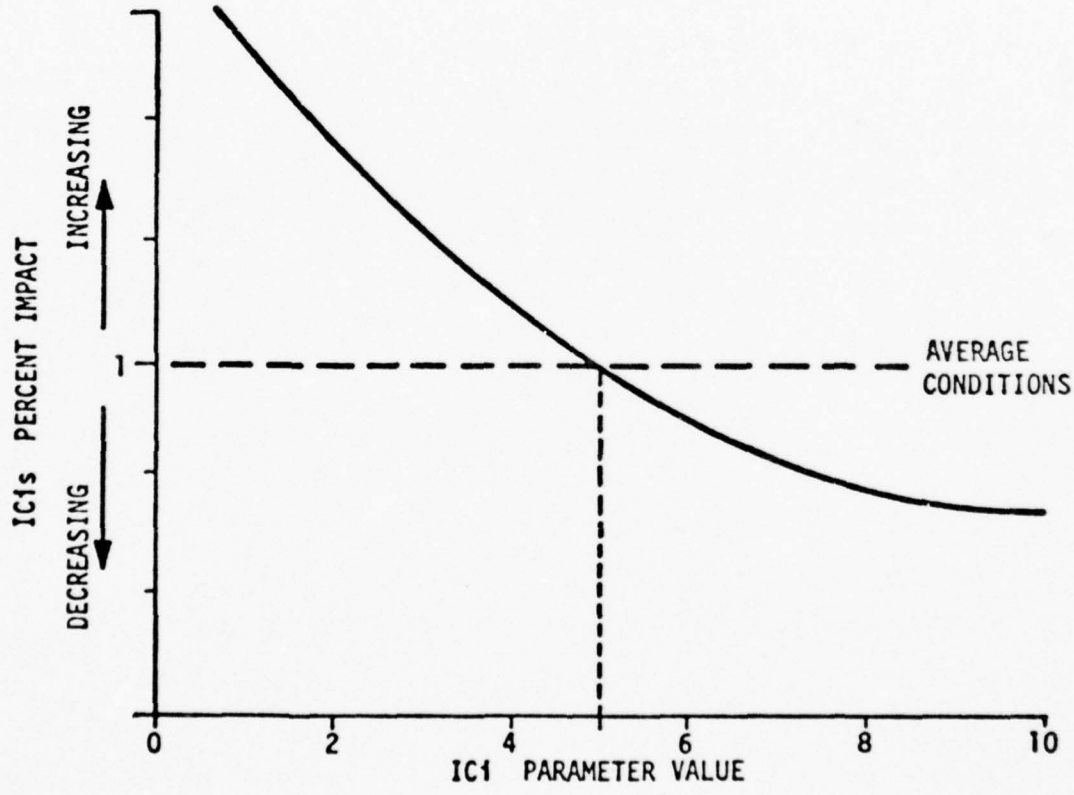


Figure 7-3 Predicted Impact Relationship

SETS is a generic name for the driver software for both the REVS generated functional and analytic simulators. Since the required SETS processing is applications and program peculiar, there are few a priori criteria for its development cost. Also, the specification and design of SETS falls in the domain of systems engineering. This occurs because it is at that level that the environment and its effects are studied/formulated in the most detail. SETS costs have, therefore, been somewhat decoupled from the overall software requirements engineering cost model and can be estimated using classical engineering and software development rules.

The Gamma models that form the REVS analytic simulator demonstrate the existence of a feasible mathematical solution. There are two separate activities involved in Gamma development: first, the mathematical formulation of the algorithms, and secondly, their implementation as PASCAL procedures. The first activity is best performed by experts in the various disciplines associated with technology or weapons systems operations. Costs associated with this activity are, therefore, provided by the individuals directly involved and are assumed to be inputs to the SRE cost estimation scheme. Algorithm implementation as RSL GAMMAs, therefore, becomes a standard software development cost estimation process.

7.2.2 Cost Model 2 (CM2)

The approach to a validated SREM cost model, as described in the previous section, is somewhat ambitious. Accurate collection of the necessary data represents one problem area. The lead time necessary to generate and calibrate the information represents a second potential constraint. An interim cost model, that is simplier than CM1 and can be used as the cost estimating baseline in the near term is presented here.

For Cost Model 2, resource expenditures for SREM applications are divided into two components.

$$\text{Cost of}_{SRE} = \text{Cost}_{\text{Functional Requirements}} + \text{Cost}_{\text{Performance Requirements}}$$

Resources for generating the functional and performance requirements are again calculated based on the total number of RSL elements that must be developed. For the development of functional requirements, the primary RSL elements are:

- ALPHAs
- R_NETS
- DATA
- INTERFACES
- MESSAGES.

Figure 7-4 presents a preliminary correlation of the results for this approach on the three projects completed to date. The dashed line represents a postulated curve fit. Note that all resource expenditures were used, including the simulation driver development cost.

Given the overall cost of the functional requirements, a finer breakdown is needed for each intermediate milestone or SRE methodology phase. Referring to Table 7.5, three logical breakdowns of the activities are indicated. At the end of the first three methodology phases, a complete description of the R_NETS, ALPHAs, etc., that constitute the functional requirements have been completed. At the end of Phase 5, the functional models have been developed and requirements testing or dynamic validation conducted. The third set of activities are miscellaneous in nature and are conducted throughout the course of the development with the exception of generation of the simulation driver. For the purpose of cost and scheduling the development process through the static validation, which occurs at the end of SREM Phase 3 requires approximately 30 percent of the total cost of generating the functional requirements. Dynamic validation requires approximately 40 percent of the total required resources. Miscellaneous activities encompass the remainder of the cost, with development of the simulation driver being the most expensive of these activities.

Given the total cost for developing functional requirements (see Figure 7-4), then the 30-40-30 percentages can be used to determine the cost for each of the three milestones within the development. A further

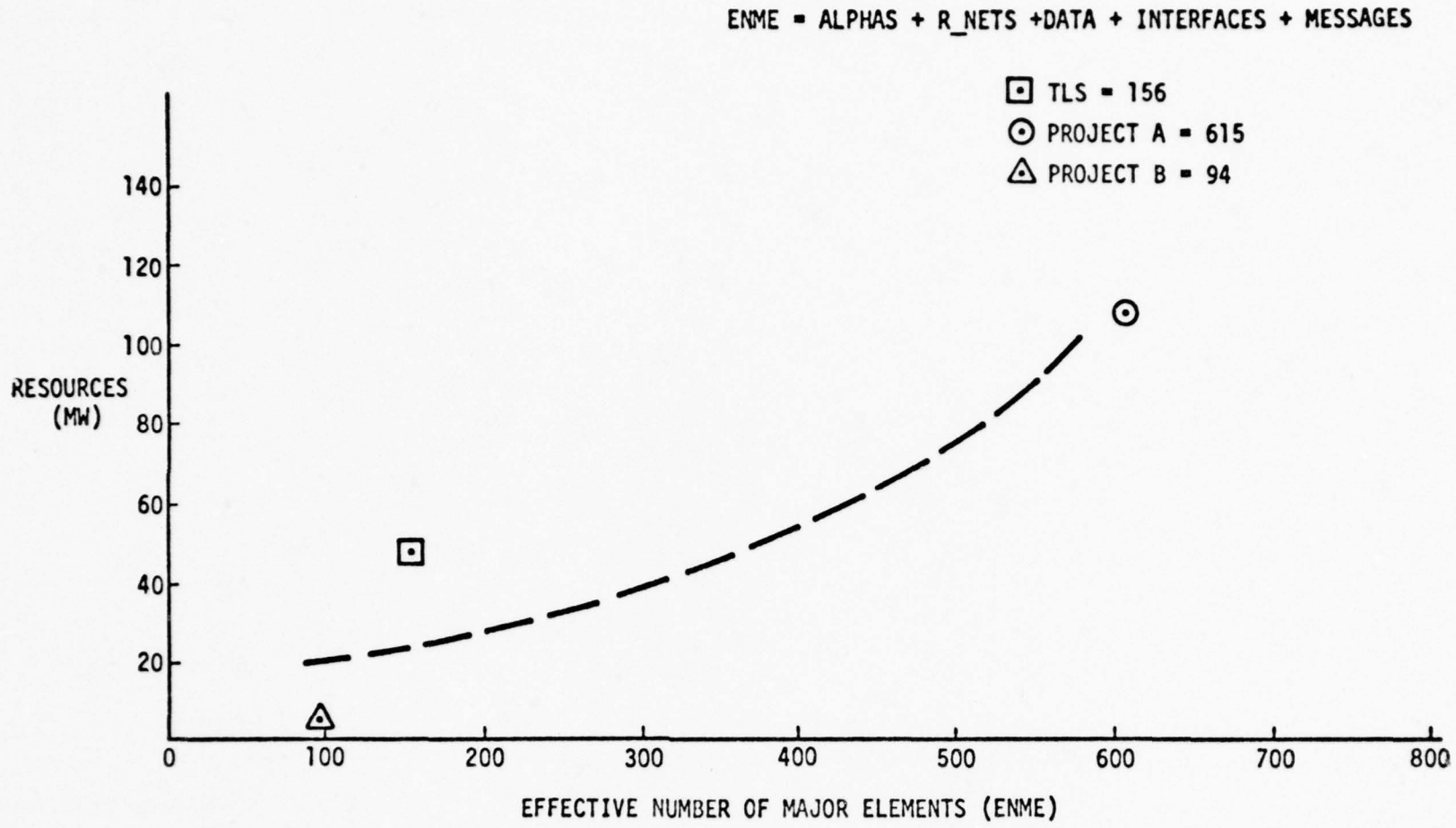


Figure 7-4 Cost Summed Over Major Elements

Table 7.5 Percentage Cost Breakdown

	TLS		PROJECT A		PROJECT B	
	MWs	%	MWs	%	MHs	%
I. Definition of Subsystem Elements	2	4.2	13	12.0	3	3.5
II. Evaluation of Kernel	5	10.3	4	3.7	8	9.3
III. Completion of Functional Description	10	20.8	15	13.9	11	12.8
M1 Static Validation	17	35.3	32	29.6	22	25.6
V. Development of Functional Models	12	25.0	22	20.4	8	9.3
Requirements Testing	6	12.5	25	23.1	24	27.9
M2 Dynamic Validation (Functional)	18	37.5	47	43.5	32	37.2
IV. Informative Material and Traceability	4	8.4	12	11.1	0	0
IV. Documentation	1	2.1	2	1.9	8	9.3
Simulation Driver	8	16.7	15	13.9	24	27.9
Misc. Activities (Subtotal)	13	27.2	29	26.9	32	37.2
TOTAL	48	100	108	100	86	100

breakdown can be formulated to meet Milestone 1, static validation. Phase 1 is basically the initial definition of the R_NETS, the connection of the Input/Output INTERFACEs and definition of the preliminary or high level ALPHAs. In Phase 2 this information is entered into the REVS data base. Then, in methodology Phase 3, the functional description is completed with all ALPHAs defined to the lowest required level and all DATA items defined. These three activities are clearly interrelated. The greater the amount of time and resources expended in Phase 1 then a corresponding smaller amount of resources must be expended in Phase 3 to complete the functional description. The SREM program manager has the discretion of applying the majority of the resources in Phase 1 or Phase 3 depending on the particular problem or application.

Required costs for development of Performance TESTs follows the same philosophy as that described above for SREM functional requirements definition. To date, detailed Performance TESTs have been generated only as a result of specific demonstration problems. None of the three projects addressed the full range of performance necessary for a complete specification. At this point, no measured data exists to assist identification of the major RSL elements that will quantify the expected cost of Performance TESTs.

8.0 MANAGEMENT CONTROL

The Software Requirements Engineering activity must interface with other activities during the system development phase. Figure 8-1 explains the major interests shared by SRE with these other activities. The key interactions may be summarized as follows:

- SRE supports SE in system definition by accepting the specifications and pointing out deficiencies.
- SRE may question the subsystem allocation because of implementation problems.
- SRE participates in the overall system cost/schedule planning and control by providing status data to SE and visibility to Process Design.
- SRE must work with the other subsystem engineering activities to define interfaces to a functional level. Process Design can work to design the detail interfaces. When a referee is needed the SE must decide interface issues.
- SRE defines the processing via software requirements which may be modified if real-time implementation is a problem.

8.1 CONTROL MECHANISMS

Management of SREM can be viewed as a two-level process. REVS provides certain direct, automatic control of the product under development (the software requirements). As described in Table 8.1, this frees management to focus on higher level issues and the implementation of REVS. With a validated REVS, implementation of its control features is completely mechanical and can be delegated with confidence. In particular, the following observations can be obtained from Table 8.1:

- At each level the manager can focus on intent, softness of decisions, level of detail and schedule performance.
- At the BETA, GAMMA and PATH/PERFORMANCE levels, cost performance becomes important. Overkill must be avoided by maintaining pressure to use the simplest, cheapest models which will do the job. (Thus, level of detail is an issue here also.)
- At the GAMMA level the manager must assess real-time feasibility even though the GAMMA models are not real-time.

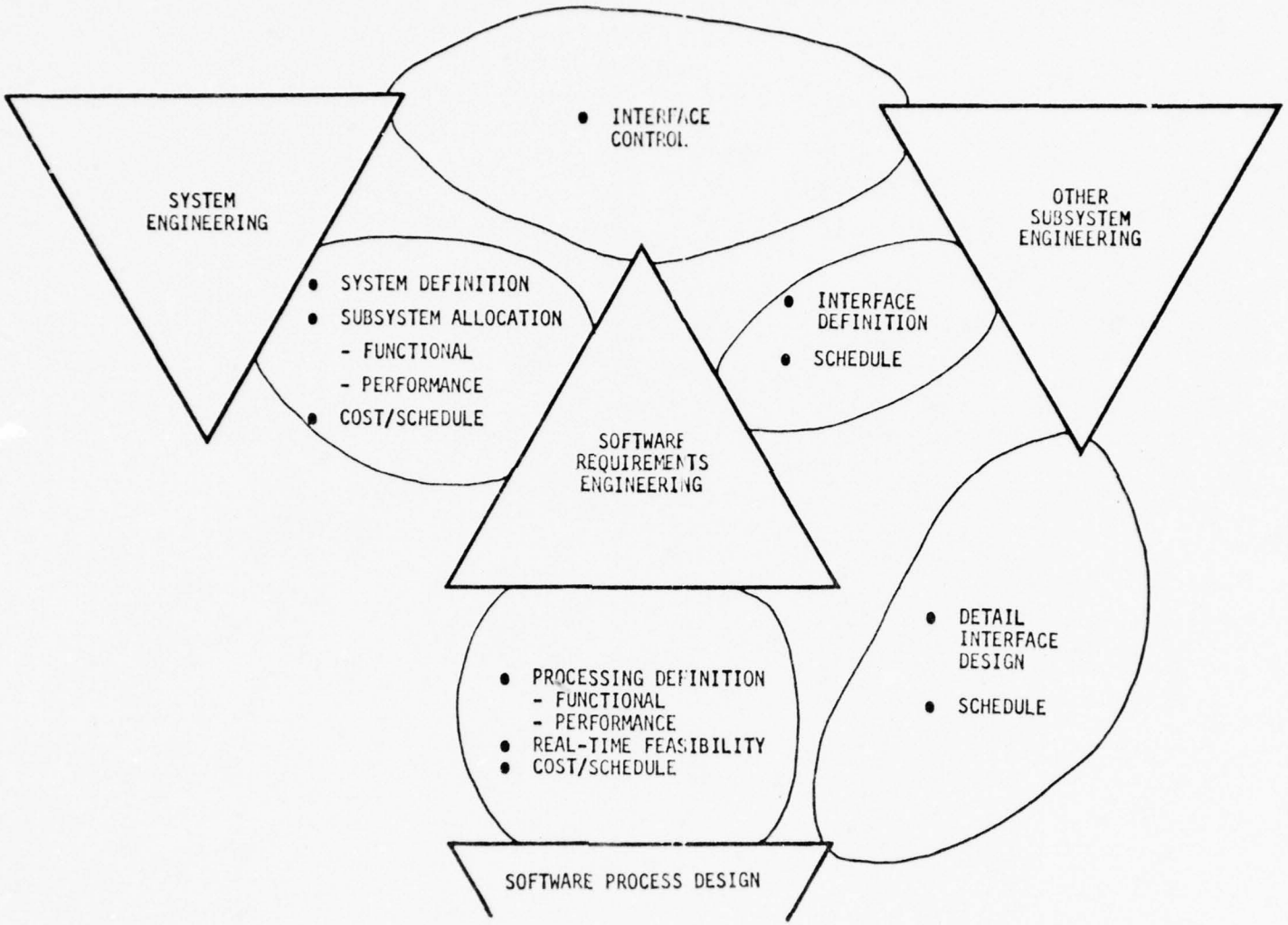


Figure 8-1 Interests Shared by SRE

Table 8.1 SREM Focus of Management Control on Substantive Issues

CONTROL PROVIDED BY REVS	MAJOR MANAGERIAL ISSUES
ALPHA: • TRACEABILITY • DECISIONS • INTERNAL CONSISTENCY - STATIC • STRUCTURAL COMPLETENESS	ALPHA: • INTENT OF SOURCE SPECS REFLECTED • SOFTNESS OF INTERIM DECISIONS • LEVEL OF DETAIL • SCHEDULE PERFORMANCE
BETA: • ALL OF THE ABOVE • INTERNAL CONSISTENCY - DYNAMIC • EXTERNAL CONSISTENCY • UPWARD CONSISTENCY WITH ALPHA LEVEL	BETA: • ALL OF THE ABOVE • COST PERFORMANCE
GAMMA: • ALL OF THE ABOVE • NON-REAL-TIME FEASIBILITY • UPWARD CONSISTENCY WITH BETA LEVEL	GAMMA: • ALL OF THE ABOVE • REAL-TIME FEASIBILITY
VALIDATION PATHS/PERFORMANCE REQUIREMENTS: • ALL OF THE ABOVE • UPWARD CONSISTENCY WITH GAMMA LEVEL	VALIDATION PATHS/PERFORMANCE REQUIREMENTS: • SAME AS BETA • IMPLEMENTATION OF REVS

- Previously, managers had to strain to get the kind of visibility easily obtained with SREM/REVS.

The major managerial issues identified in Table 8.1 require mechanisms outside of REVS. These are the more widely used managerial controls, and are much more effective when based on the timely, complete, and organized information provided by REVS. The kinds of control mechanisms used for these issues are shown in Table 8.2 and are summarized below:

- Internal reviews can be used to establish confidence in the software requirements by addressing items checked in the Table.
- Starting with a parametric cost model, projected cost can be estimated as actual parameter values are determined (e.g., number R_NETs, ALPHAs, etc.).
- Earned value as a measure of progress toward each milestone can be used in the C-SPEC sense to evaluate cost/schedule performance in a continuous fashion. Earned value can be quantified by using number of ALPHAs, BETAs, GAMMAs, PATHs, TESTs, etc. weighted by complexity factors.
- The milestone schedule is a standard schedule performance control best implemented by public display.
- Reviews with the system engineer should help understand questions of intent and softness of decisions plus adequacy of level of detail.
- The process designer should be involved in reviews of level of detail and real-time feasibility (includes storage capacity).

8.2 CHANGE CONTROL

The process of change control is a critical one on a large, complex system development activity. Figure 8-2 shows how collected decisions can be folded into the baseline as scheduled updates. The updates are inserted at points in the schedule where they are likely to have significant inputs from within the SRE activity and from System Engineering and Process Design. They also follow specific measures of the software requirements, namely the initial dynamic evaluation and the initial feasibility demonstration.

Table 8.2 Control Mechanisms for Major Managerial Control Issues

MAJOR MANAGERIAL CONTROL ISSUES	CONTROL MECHANISM					
	INTERNAL REVIEWS	COST PARAMETER MEASUREMENT	EARNED VALUE	MILESTONE SCHEDULE	SYSTEM ENGINEER REVIEW	PROCESS DESIGN REVIEW
SOURCE SPEC INTENT	X				X	
DECISION SOFTNESS	X				X	
LEVEL OF DETAIL	X				X	X
COST PERFORMANCE		X	X			
SCHEDULE PERFORMANCE			X	X		
REAL-TIME FEASIBILITY	X					X

MILESTONE / DECISIONS RELATED TO	INITIAL R-NETS DEVELOPED (Phase I)	STATIC EVALUATION (Phase 3)	DYNAMIC FUNCTIONAL EVALUATION (Phase 5)	DYNAMIC PERFORMANCE ASSESSMENT (Phase 8)
SOURCE SPECIFICATION	X	X	X	X
SOFTWARE REQ. (ALPHA LEVEL)		X	X	X
BETA LEVEL			X	X
GAMMA LEVEL				X

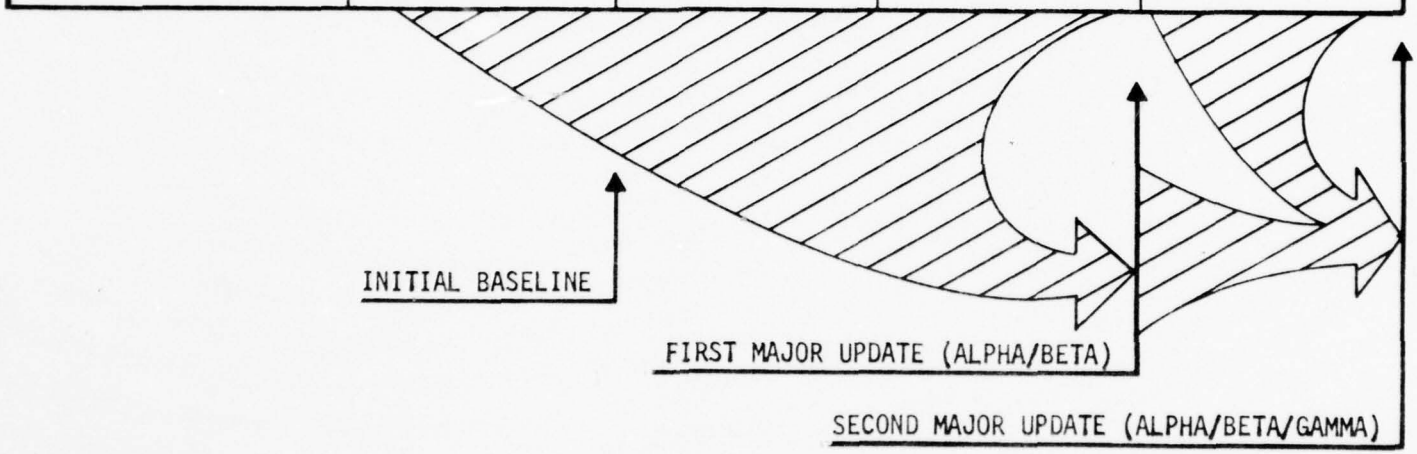


Figure 8-2 Sample Change Flow into Update of Baseline

8.3 ACCEPTANCE OF THE SOFTWARE REQUIREMENTS

Both the System Engineer and Process Designer must accept the end product software requirements. SREM possesses key features which should assist the SRE manager in effecting this buy-off. These are explained in Table 8.3 in terms of answers to concerns of both System Engineering and Process Design.

Table 8.3 Acceptance of the Software Requirements

SYSTEM ENGINEER CONCERNS	SRE ANSWER	PROCESS DESIGNER CONCERNS
1. DO SOFTWARE REQUIREMENTS FAITHFULLY TRANSLATE SOURCE SPECS?	• REVS TRACEABILITY • INTENT REVIEWS WITH SYSTEM ENGINEER • QUICK RESPONSE • CONTROLLED BASELINE	
2. WHAT CHANGES IN SOURCE SPECS ARE NECESSARY FOR IMPLEMENTATION?	• DOCUMENTED DECISIONS • STATIC EVALUATION • DYNAMIC EVALUATION • CONTROLLED BASELINE	
3. WHAT IS COST/SCHEDULE DELTA FOR A SYSTEM CHANGE?	• QUICK RESPONSE TO DEFINE SOFTWARE REQUIREMENTS CHANGES FOR COST/SCHEDULE EVALUATION	
	• SRE METHODOLOGY • LEVEL OF DETAIL REVIEWS	1. IS LEVEL OF DETAIL TOO CONSTRAINING? ADEQUATE?
	• RSL • REVS	2. REQUIREMENTS CLEAR, COMPLETE AND CONSISTENT?
	• FEASIBILITY DEMO • REAL-TIME REVIEWS WITH PROCESS DESIGNER	3. IMPLEMENTATION FEASIBLE?
	• REVS TEST DEFINITION	4. WHAT IS SOFTWARE ACCEPTANCE CRITERIA?
	• BASELINE CONTROL • DECISION TRACE	5. ARE REQUIREMENTS FIRM?

9.0 CONCLUSIONS

This manual has explained both the technical and management considerations in the development and validation of software requirements using the tools and techniques of SREM. The engineering and management principles explained here are not actually new -- they are the result of hundreds of man-years of experience and observation of large-scale, real-time software development. The discipline of RSL and the power of REVS, however, are new, as is the formalization of the detailed steps to be followed in their use.

This manual will not make instant engineers or managers out of inexperienced people. It will, however, guide the experienced engineer and manager in the application of RSL, REVS, and SREM techniques to obtain a software requirements specification which is superior in terms of the basic qualities of a good requirements specification.

10.0 APPLICABLE DOCUMENTS

1. Boehm, B. W., "Software Engineering", IEEE Transactions on Computers, December 1976.

2. "Software Requirements Engineering Methodology", Final Report, TRW Report No. 27332-6921-019, 12 December 1975.

3. Bell, T. E., and Thayer, T. A., "Software Requirements: Are They Really a Problem?", TRW Software Series, TRW-SS-76-04, July 1976.

4. "Information Processing/Data Automation Implications of Air Force Command and Control Requirements in the 1980's (CCIP-85)", SAMSO Report XRS-71-1, April 1972.

5. "Current Software Requirements Engineering Technology", TRW Report No. 22944-6921-014, August 1974.

6. "Software Requirements Engineering Methodology Notebook", TRW Report No. 22944-6921-020, October 1974.

APPENDIX A

SAMPLE DPSPR - X1 TRACK LOOP EXPERIMENT

I. INTRODUCTION

1.0 PURPOSE AND SCOPE

This report presents the preliminary Data Processing Subsystem Performance Requirements (DPSPR) for the Track Loop Experiment (X-1). The goals for this experiment are presented in Section 2.0.

The primary intent of this document is to provide the initiating input to the Software Requirements Engineering Methodology which will subsequently produce a Process Performance Requirements (PPR) specification for the Track Loop System Data Processing Subsystem. It is further the intent of this document to present an example of the required contents and level of detail of a DPSPR discussed in Reference 1. As such, this example will provide a more concrete basis for technical exchange between the Software Requirements Engineering, DPSPR and V&V contractors. Such interchange is considered necessary to arrive at a final definition of the required form, contents and format of a DPSPR. Part II of this report constitutes the example DPSPR.

1.1 The Track Loop System

The Track Loop System (TLS) is a subset of a Preliminary Ballistic Missile Defense System which is capable of nearly autonomous execution in response to external stimuli. It is the simplest known subsystem with properties of interest for software definition, and it is one which has been studied extensively, both in the academic literature and in such practical programs as Site Defense. Therefore, it has been selected as the testbed for supporting experimentation in development of the methodology for software requirements. A pictorial representation of the TLS is provided in Figure A-1.

1.1.1 Preliminary Ballistic Missile Defense System

A Preliminary Ballistic Missile Defense System (PBMDS) has been postulated as an environment in which the TLS would execute. It is a generalized representative of the class of systems currently in development, and is particularized for the TLS through representative but non-real specifications where required.

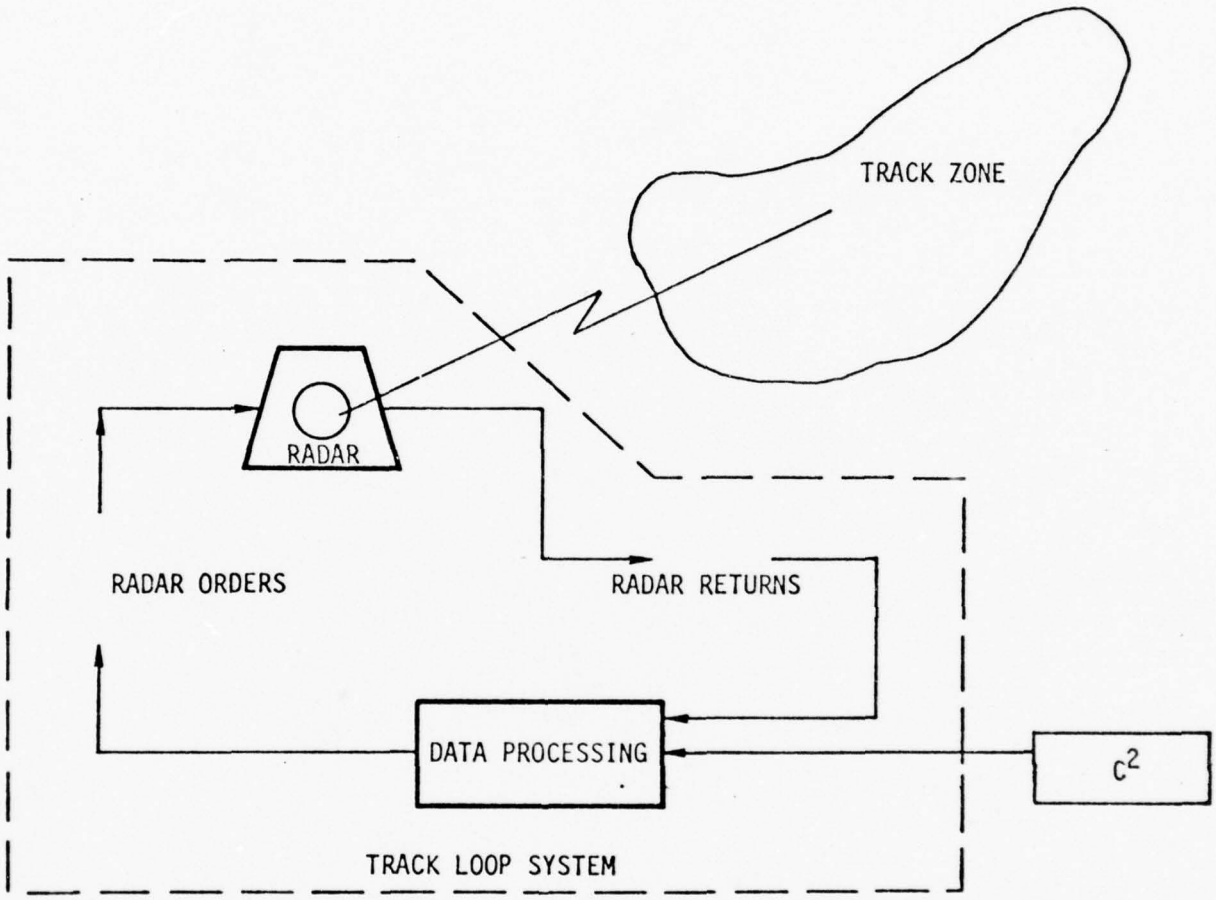


Figure A-1 Track Loop System

The top level flow of the PBMDS is shown in the functional flow block diagram, Figure A-2. In the Conduct Engagement mode, an object entering the search region will be detected and designated, tracked, discriminated, and engaged (as required) in defense of the ground facilities. Those functions are implemented through the Data Processing System (DPS), a radar or other sensor, and a means of neutralizing hostile objects. For the purpose of the TLS, only the radar need be defined in detail; other system elements are identified only to the extent that they impact DPS requirements.

1.1.2 TLS Requirements

Functional Requirements on the TLS would normally be contained in a system specification (Level 1, or A in MIL STD 490 terminology). If software is developed from the requirements provided in Section II of this report, and if that software is to be installed and exercised in the field, then such a system specification may be required. In the interests of both economy and timeliness, only the subsection of the A Specification required for the DPSPR is provided here.

1.1.2.1 Initialization

The TLS shall accept C^2 messages for initialization with the following properties:

a) An estimate of state shall be generated external to TLS and forwarded to TLS over the communication channel to initiate tracking.

b) If an object corresponds to that estimate, the estimation accuracy shall be such that the expected (1-sigma) deviation of the object from a perfect extrapolation of state shall be in accord with Table 1-1.

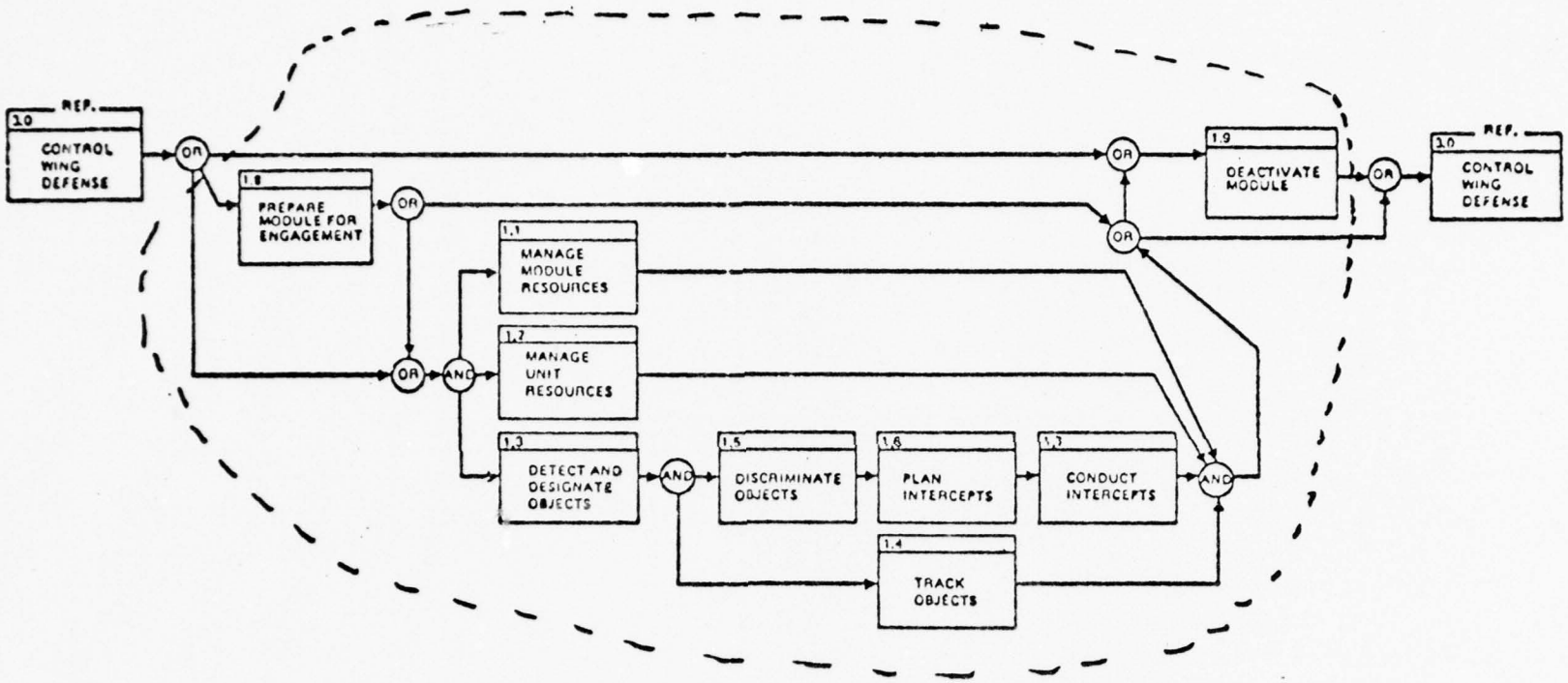
c) Initialization estimates shall be provided (handed off) at a rate not exceeding 150 per second over any interval greater than 50 milliseconds.

d) The total number of handoffs shall not exceed 1500.

e) The total number of real objects shall not exceed 300.

f) A handoff may be an estimate on a real object or an estimate relating to a state at which no object is located (ghost). Multiple handoffs of a single object may be generated.

g) Each handoff shall consist of a unique designator, the state vector and its covariance matrix.



NOTES:
1. 1.1 AND 1.2 PROVIDE DIRECTION TO AND RECEIVE INPUTS FROM 1.3, 1.4, 1.5, 1.6, AND 1.7 THROUGHOUT THE ENGAGEMENT.
2. FURTHER FEEDBACKS AND INTERCONNECTIONS ARE SHOWN AT LOWER LEVELS WHERE THEY EXIST.
3. ORDER SHOWN DOES NOT IMPLY THAT FUNCTION ENDS WHEN FOLLOWING FUNCTIONS BEGIN

Figure A-2 Conduct Engagement Function (PBMDS Function 1.0)

BEST AVAILABLE COPY

Table A.2 Table A.1 Handover Errors in Radar Face Coordinates

COORDINATE	MINIMUM	MAXIMUM	UNITS
R	3	5	M
U	0.4	0.6	msine
V	0.03	0.05	msine
$\dot{R}$	40	55	m/sec
$\dot{U}$	2	4	msine/sec
$\dot{V}$	2	4	msine/sec

NOTES:

1. All data $1\text{-}\sigma$
2. Handover altitude $\geq$ 65 K meters

1.1.2.2 Termination

a) Redundant images of objects shall be dropped from track in order to conserve radar resources. The probability of dropping track on a non-redundant image shall be considered in determining leakage.

b) Ghosts shall be dropped from track in order to conserve radar resources. The probability that a non-redundant image is dropped as a ghost shall be considered in determining leakage.

c) For flight safety, no track pulse shall be commanded with true elevation angle less than 3°.

d) Track shall be dropped in response to an external command representing handoff to another defense facility or successful intercept. No track pulse shall be transmitted to a designated image more than 100 milliseconds after such a command appears at the TLS port, with probability .3.

1.1.2.3 Tracking

a) The TLS shall generate state estimates sufficient to support discrimination through beta estimation accuracy in accordance with Figure A-4 and impact point prediction in accordance with Figure A-5.

Beta is required in the system for a variety of purposes at different stages in the engagement of an object. Early in track, it is used for junk rejection; at an intermediate state, it forms a key element of discrimination in elimination of decoys and assessment of the danger imposed by an RV; shortly thereafter, it is essential to intercept planning in estimating the intercept point. The needs overlap in practice, so that a common ordinate for the plot of accuracy requirements is needed. But each of the aspects of use of beta dictates a different ordinate at the system level. The ordinate selected in Figure A-4 is time in track, a parameter known to be useful to the Software Requirements Engineer, and as useful for the systems-level definition as any of the other choices.

Similarly, impact point prediction is used initially to reject cross traffic, then to assess the threat posed by an RV; in some schema, it also assists discrimination. The first might be expressed by the system engineer in terms of accuracy against track energy; the second in terms of time to commit contour (which is in turn a function of RV type, intercept capabilities, and other parameters). Again, time in track was chosen as a convenient ordinate for the error requirements in Figure 1-5.

b) The TLS shall generate state estimates sufficient to support object intercept in accordance with Figure 1-6 (TBD).

c) Leakage is here defined to be the probability that all images of a real object entered at an altitude not less than 150K feet are dropped from track for any reason other than the minimum-elevation constraints or external command defined in 1.1.2.2. Leakage in TLS shall not exceed .03.

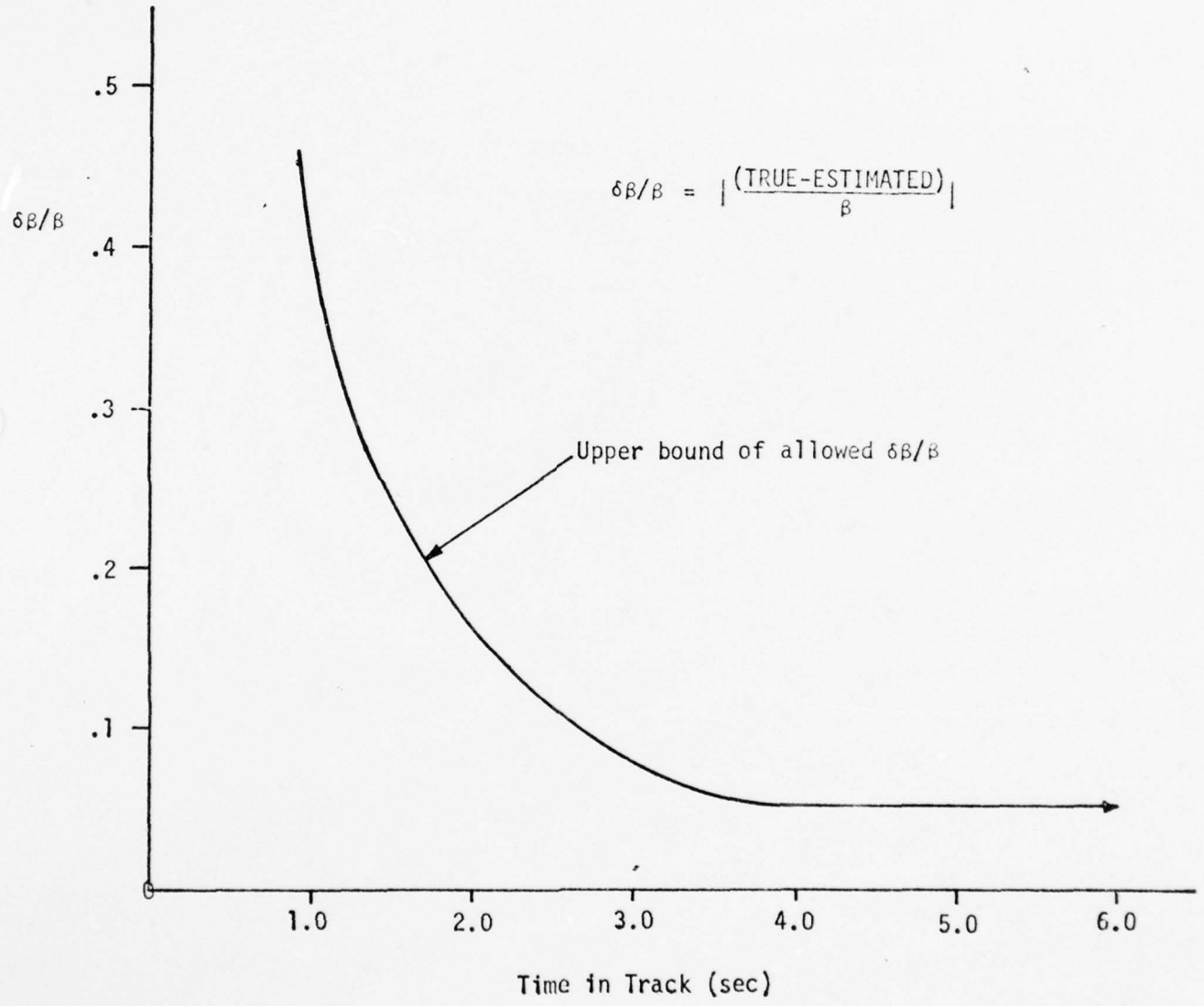


Figure A-3 Beta Error Requirements

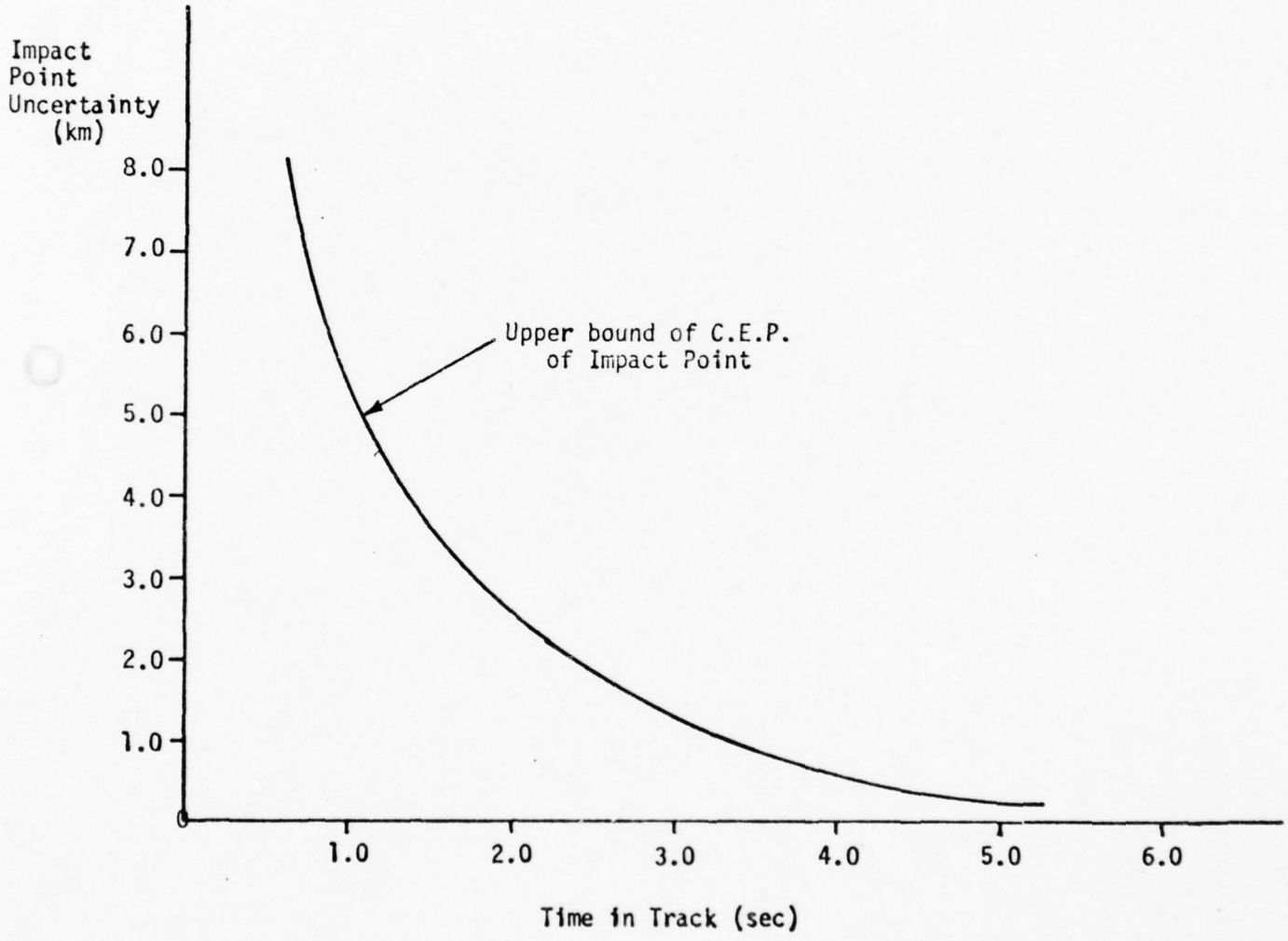


Figure A-4 Impact Point Error Requirements

1.1.2.4 Resource Control

a) The TLS shall commit to tracking functions not more than 1800 radar pulses per second, using not more than 25 kilowatts ERP.

b) The TLS shall commit to tracking functions not more than 2500 joules per object.

1.1.2.5 Data Recording

The TLS shall provide records for post test analysis of the following data:

a) Time of handoff and designator and estimate received.

b) Time of termination of track on each designation with reason for termination.

c) At intervals not greater than 100 milliseconds, the estimate of state for each designation received and not yet terminated. That estimate shall consist of the following data: position, velocity, a beta estimation parameter and estimates of the uncertainty of each.

d) At intervals not greater than 300 milliseconds, the usage of radar resources. That measure shall include: radar energy profile and DPS estimates of expected and maximum energy commanded.

2.0 SREP OBJECTIVES FOR X-1

a) Demonstrate the scope, contents and level of detail of the DPSPR, described in Reference 1, and transmit the results to the DPSPR contractors for their evaluation.

b) Demonstrate the scope, contents, format, and level of detail of PPR Volume I specified in Reference 2. The PPR will be written in preliminary RSL (see Reference 4). In particular:

   1) Evaluate the format of the PPR described in Reference 2 for completeness and adequacy.

   2) Evaluate the adequacy of the preliminary RSL definition for stating requirements.

   3) Provide the PPR for evaluation; in particular, the form and content of the performance requirements.

   4) Evaluate the extent of the traceability of the PPR to the DPSPR.

c) Evaluate the steps of SREM (see Reference 3) from the DPSPR to a PPR; in particular, the procedural steps, form, and content of the performance allocation activities.

d) Exercise the available experimental software to generate portions of the PPR (e.g., the structure segment of RSL).

3.0 LIMITATIONS

Two primary forces limit the application of the DPSPR of Section II. The first is the result of excising the track loop from the total PBMDS. Although the loop is nearly self-contained, it has extensive interfaces both in terms of requirements and in the sense of implementation with other software functions. Consequently, requirements are imposed on TLS which originate or terminate within the DPS in ways and to an extent not believed appropriate for a true, functional system.

The second major area of limitation arises from the novelty of the approach. While the methodology employed is believed valid, it has not yet been tried. (In fact, this experiment is its trial.) Consequently, it is likely that this initial DPSPR will be found to be incomplete in some materials needed for the PPR and for software development. Every effort has been made to anticipate such failings, but we anticipate methodologic bugs in the same way we would expect to find bugs in a new compiler or other major software development.

3.1 System Objectives

Isolation of TLS from PBMDS was artificial, and the objectives of Section 1.1 are correspondingly less than real. The selection of those objectives was based on simplicity of implementation, expected usefulness of the TLS as a testbed, and availability of comparison criteria. Since no DPSPR had ever been prepared before, it was not possible to select criteria to optimize its quality. The objectives provided are believed to be good for generation of the DPSPR, but cannot be shown to be optimal.

3.2 Allocation to DP

Again, the absence of precedent and the artificiality of the TLS make an "optimum" allocation of functions unreasonable. A complete allocation is provided, which is believed to be sufficient for the experiment.

3.3 Level of Detail

The objective of the DPSPR of Part II is to provide the complete specification required for the PPR Feasibility Demonstration. It is likely

that some of the material will prove to be unnecessary; it is certain that not all data presently missing will be required before initiating work on Volume I of the PPR. Therefore, revisions are planned both to complete specification of the requirements and to delete data found to be non-essential.

3.4 Formulation of DP Requirements

The X-1 DPSPR defines some data in a form and to a depth we feel to be undesirable. In particular, efforts are now under way to express requirements on Redundant Track Elimination in terms of total radar energy per object. Converting both redundant tracking and other explicit requirements to a derivable form will be an objective for revisions of the DPSPR.

3.5 Corollary Documents

The DPSPR is one document in a family required to develop the Process Performance Requirements (PPR). For the TLS, the other documents are:

- TLS System Requirements
- TLS Radar/DPS Interface Specification
- TLS C^2/DPS Interface Specification
- TLS Radar Performance Specification*
- TLS Environment and Threat Model Definition.*

The system requirements are sketched in Section 1.1 above. The starred (*) documents in this family are not yet prepared.

For the purposes of development of the PPR, the absence of the documents is believed to be less than critical. For the X-1 effort, the contents of the missing specifications are subsets of the corresponding Terminal Defense Program (TDP) documents. Therefore, the required information for the PPR is available from TDP documentation. However, a property of the methodology is its recognition that not all specifications ultimately required to support software design will be available early in that design effort. To the extent that the TLS documentation corresponding to initiation of a PPR is required, the TDP documents are excessive. Thus, there is an effort required as a part of the ongoing work to edit the TDP material to the specific, appropriate contents for the stage of development represented by the DPSPR. That work is under way on the Radar/DPS Interface specification, and will soon be undertaken on the other elements of the DPSPR package.

4.0 REFERENCES

1. TRW Report 27332-6921-003, "Software Performance Requirements - DPSPR Content Requirements", (CDRL A004), Revision 1, December 12, 1975.

2. TRW Report 27332-6921-004, "Software Performance Requirements - PPR Content Requirements", (CDRL A005), Revision 1, December 12, 1975.

3. TRW Report 27332-6921-005, "Software Requirements Engineering Methodology Description Special Report", (CDRL A00C), Revision 1, December 12, 1975.

4. TRW IOC TEB-75-6000.02-135, "Requirements Statement Language Descriptions", 1 May 1975.

II. DATA PROCESSING SUBSYSTEM
PERFORMANCE REQUIREMENTS - TRACK LOOP SYSTEM

1.0 SCOPE

This specification establishes the functional, performance, design and test requirements for the Track Loop System (TLS) to the extent necessary to develop the Process Performance Requirements (PPR) for the Data Processing Subsystem (DPS) portion of the system. The TLS is a test configuration of an integral element of a Preliminary Ballistic Missile Defense System (PBMDS).

The primary objective of this specification is to constrain the DPS so that it fulfills the intended obligations to the functions and performance of the overall TLS of which it is a part. To accomplish this, this specification

- identifies the TLS mission and performance goals.
- identifies the various subsystems, their functional capabilities, and the interfaces between the Data Processing Subsystem and each of the others.
- allocates a portion of the system performance to the Data Processing Subsystem.
- describes the system operational design, i.e., how the subsystems are to be used in order to achieve the system performance goals by utilization of the subsystem capabilities.

This specification provides the technical basis for the development of the DPS, but is not a TLS system specification.

2.0 APPLICABLE DOCUMENTS

2.1 DPSPR Content Requirements, TRW Report 27332-6921-003, (CDRL A004), Revision 1, December 12, 1975.

2.2 TLS System Requirements, (Part I, Section 1.1 of this report).

2.3 TLS Radar/DPS Interface Specification TRW Report No. 27332-6921-012

2.4 TLS C^2/DPS Interface Specification TRW Report No. 27332-6921-013

2.5 TLS Radar Performance Specification

2.6 TLS Environment and Threat Model Definition

3.0 DATA PROCESSING SUBSYSTEM

The TLS is defined in [2.2] (Part I, Section 1.1 of this report). Section 1.1.2 provides the system requirements, while Figure A-1 represents the operation of the system. TLS consists of a radar and a data processor, and receives input data from radar echoes and from interface with the C^2 system. A model of the system environment is to be provided as [2.6]. Within the TLS, the radar-data processor interface specification is to be [2.4], while the C^2/DPS interface will be defined in [2.3]. The radar models will be in [2.5].

The TLS DPS has been allocated requirements from those established on the TLS as a whole. The resulting requirements on the DPS are defined in Section 3.2 of this DPSPR. Traceability of the DPSPR requirements to the TLS requirements is shown in Figure A-1. All TLS requirements not allocated to the DPS are satisfied by the radar. Quantitative verification of the sufficiency of the allocation will be undertaken in the preparation of the PPR.

3.1 Traceability

Figure A-1 is the traceability matrix for this DPSPR, and illustrates the relationship between the TLS requirements [2.2], and the DPS requirements of Section 3.2. It also depicts the allocation of portions of system requirements to the radar; not all of that allocation is clearly visible in the interface specification, some of it being located in [2.5].

The function of the traceability matrix is twofold: to locate the subsystem requirements derived from each system requirement, and to facilitate recognition that each DPS requirement originates from an appropriate source. The first function is insurance that the DPSPR (and its counterparts for other subsystems) are sufficient to embody the system requirements, while the second verifies that no gratuitous requirements have been introduced.

To determine the source(s) of a DPS requirement, the appropriate row is located in the left-hand column. Reading across the row, an entry of "C" indicates that virtually complete satisfaction of the system requirement for that column is provided. A "P" indicates that a portion of the system requirement is there satisfied. Scanning the entire chart, one finds that the DPS

PART I 1.1.2 / PART II 3.2		.1							.2					.3			.4		.5				
		a	b	c	d	e	f	g	a	b	c	d	e	a	b	c	a	b	1	2	3	4	5
.1		C	C	C	C	C	C	C															
.2	a								P								P	P					
	b									P							P	P					
	c										C												
	d										P		C				P	P					
	e											C					P	P					
.3	a													P	P								
	b															P							
	c															P							
	d													P	P								
	e																						
	f													P	P	P							
	g													P	P	P							
.4	a																P	P				P	
	b																P	P					
.5	1																		C				
	2																			C			
	3																				P		
	4																					P	
RADAR									P	P	P			P	P	P					P		P

Figure A-1 Requirements Traceability Matrix

requirement to satisfy the interface specification is not traceable to the system requirements; however, it is clearly required, and is clearly inappropriate for the system specification.

Determining the impact of a system requirement on the subsystems is constructive both in confirming that every requirement is covered and in tracking the consequences of a change to the system specification. Reading a column will show the partial (P) or complete (C) satisfaction of a system requirement in the corresponding DPSPR section. In the present case, each system requirement has some DPS impact; that might not be true in general. For example, if a hard stop on elevation angle were implemented in the radar, then the third requirement under 3.2.2 would be deleted, and the implementation of the third requirement of (Part I) 1.1.2.2 would be virtually completely contained in the radar subsystem.

3.2 Data Processing Subsystem Requirements

Many of the following specific requirements are stated in terms of a value and a probability of its satisfaction. In each such case, the interpretation shall be that at every point in the engagement, the probability of satisfying the inequality shall be at least the stated value. The required confidence in that assertion is established in test planning.

3.2.1 Initialization

a) The DPS shall accept commands from the C^2 to initiate track on a designated object. A radar order in response to such a command shall be provided at the radar interface for transmission within 55 milliseconds of the command with probability .9. These commands are defined in Reference 2.4 [TLS C^2/DPS Interface Specification].

b) Capacities. The DPS shall be able to accept handoffs at a rate of 150 per second over any interval greater than 50 milliseconds, and a total of 1500 handoffs per engagement, of which up to 300 will be real images.

3.2.2 Termination

a) The DPS shall command not more than 15 pulses to a redundant image with probability .7, nor more than 50 with probability .99.

b) The DPS shall command not more than seven pulses to a ghost with probability .7.

c) The DPS shall command no track pulse with elevation angle less than 3°.

d) The DPS shall command no track pulse to an image of an object which has achieved an elevation of less than 5°, or which will attain such an elevation by the time of transmission, with probability .9.

e) For an image on which a drop-track command is received, the DPS shall command no transmission with execution time more than 100 milliseconds after appearance of the order at the C^2 interface with probability .7, nor more than 300 milliseconds with probability .99.

3.2.3 Tracking

a) The DPS shall maintain an estimate of state for each image in track. Defining the true state of an image by Appendix A, estimated state shall deviate from true state by not more than the tolerance of Figure A-1 (TBD) with the probabilities of that Figure, where the assessment is relative to the time of the last processed return.

b) The probability that all images of an object shall be dropped as redundant shall not exceed .01.

c) The probability that any image of an object shall be dropped as a ghost shall not exceed .2.

d) The DPS shall command track pulses at a rate sufficient to keep the propagated error defined by Appendix B less than the tolerances of Figure 3-2 (TBD) with probabilities defined in that Figure.

e) The DPS shall provide orders to the radar in accordance with the interface specification. The relevant fields, timing, etc., are defined in [2.3]. In particular, the DPS shall select waveforms, frequencies, beamwidths and related parameters in accordance with radar constraints in order to satisfy TLS performance requirements.

f) The DPS shall accept radar returns in accordance with the Interface Specification [2.3].

g) The DPS shall maintain track on each image until one of the following conditions is satisfied:

1) Drop Track command received,

2) Image determined to be redundant,

3) Image determined to be a ghost, or

4) Estimated elevation of object or of image expected to violate elevation-angle constraint before next assessment.

3.2.4 Resource Control

a) The DPS shall maintain an estimate of radar energy scheduled by track functions which shall be within three percent of the energy nominally expended, and an upper-bound estimate such that the actual ERP and total radiated energy do not exceed the bound with probability (TBD).

b) The DPS shall allocate radar commands so that not more than (TBD) joules are commanded per image, nor more than (TBD) kilowatts or (TBD) pulses/second for all images in track.

3.2.5 Data Recording

The DPS shall output to permanent file the following data:

a) Time of handoff and designation and estimate provided.

b) Time of termination of track on each designation with reason for termination. The reason shall be one of the following.

1) Drop Track Command
2) Minimum Elevation
3) Image Declared Redundant
4) Image Declared Ghost.

c) Subsequent to each state update, the resulting estimate of state on that image. Contents of that estimate are TBD.

d) At intervals not greater than 300 milliseconds the usage of radar resources. That estimate shall include the nominal and upper bounds defined in 3.2.4, and TBD additional data.

4.0 GLOSSARY

State Vector: A set of data pertaining to a specified image defining its location in space and permitting extrapolation of its location to other times. The contents of the state vector may vary with different applications; in particular, the coordinate system employed is dependent on the use to which it will be put. A conventional state vector in RFCC is: $R, U, V, \dot{R}, \dot{U}, \dot{V}, \beta^{-1}$, and the time to which all are referenced.

Object: A physical entity external to the DPS with radar reflection properties corresponding to a reentry vehicle of the defined threat.

Image: A target defined to the TLS DPS for tracking and categorization as the image to be tracked, a redundant image, or a ghost.

Ghost: An image with which no object is correlated.

Redundant Image: An image which correlates with both an object and another image. Of the set of redundant images of an object, one is intended to be retained in track while the others are categorized as redundant and dropped.

Handoff: The receipt by the TLS DPS of a valid order to initiate track on an image.

Track Termination: The receipt by the TLS DPS of a valid order to Drop Track, or the determination by the DPS that tracking should be terminated.

APPENDIX A

STATE VECTOR PREDICTION

The filter maintains the state vector in the RFCC system at all times. Specific details of the equations of motion and the method of trajectory integration employed for state vector prediction in the RFCC system are presented in this section. Figure 8-1 defines the RFCC system.

1.1 Equations of Motion

In the general formulation of the equations of motion, it is assumed that all vector variables appearing in the differential equations are appropriately referenced to the RFCC system associated with the face of a given radar under consideration. Explicit expressions for the transformed variables are given whenever they first appear.

Let the RFCC position vector to a target, denoted $\bar{r}_f$, have Cartesian components X_f, Y_f and Z_f. The differential equations describing the motion of a target in the rotating RFCC system may, in general, be written as

$$\ddot{\bar{r}}_f = -\frac{\mu \bar{R}_f}{|\bar{R}_f|^3} - \frac{\rho g \lambda}{2} |\dot{\bar{r}}_f| \dot{\bar{r}}_f - 2\bar{\omega}_f \times \dot{\bar{r}}_f - \bar{\omega}_f \times (\bar{\omega}_f \times \bar{R}_f) \tag{A.1}$$

and assuming constant λ model

$$\dot{\lambda} = 0$$

where

$\mu = GM$

G = universal gravitational constant

M = mass of the earth

$\bar{R}_f$ is the vector from the geocenter to the target

ρ is the atmospheric density which is a function of the altitude h

g = earth's sea level gravity

λ is the inverse of the ballistic coefficient

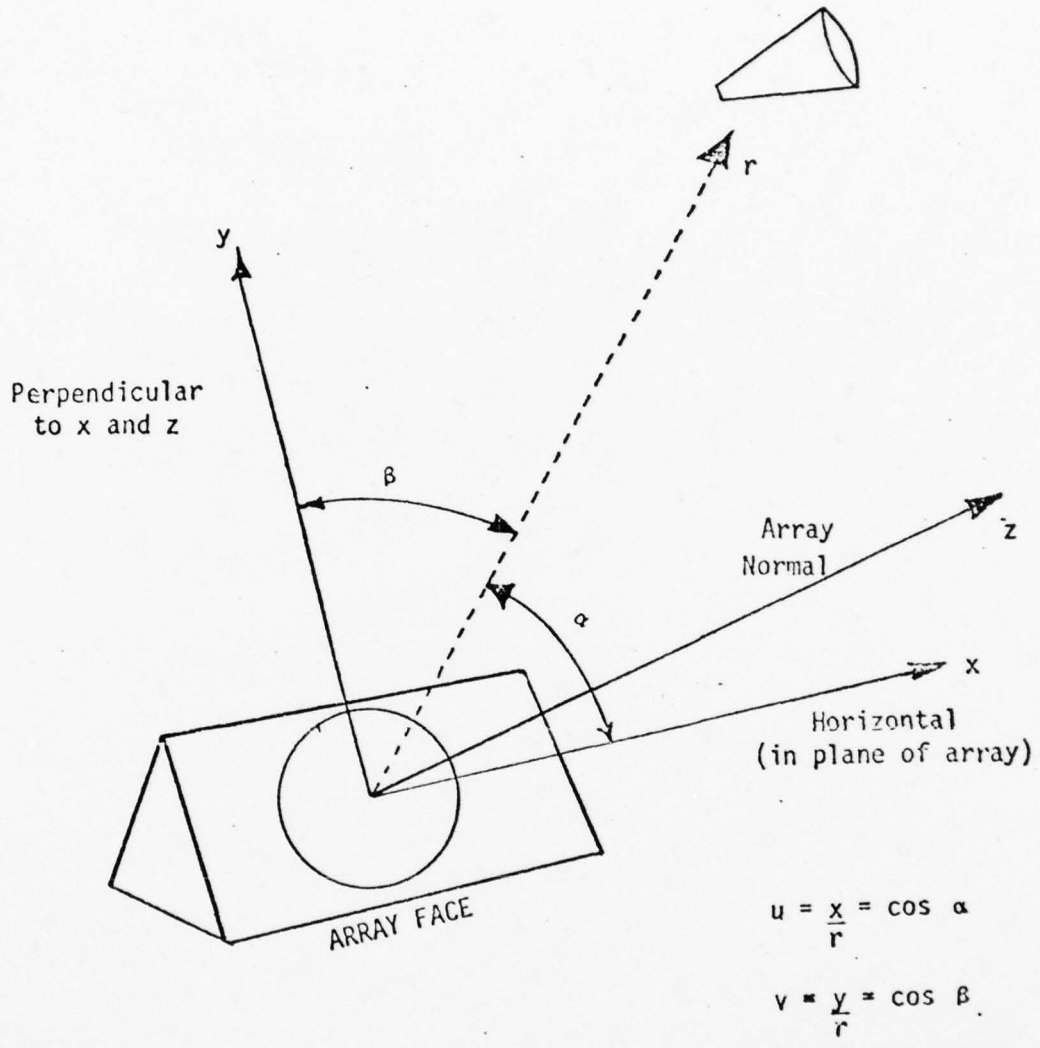


Figure A-1 Radar Face Centered Coordinates

$\dot{\bar{r}}_f$ is the velocity vector of the target

x is the vector cross-product

$\bar{\omega}_f$ earth's angular rotation rate, a vector quantity resolved along an earth-centered Cartesian coordinate system aligned with the RFCC system.

Define $\bar{R}_{sf}$ as the vector from the geocenter to the radar site expressed in the RFCC system. Then

$$\bar{R}_f = \bar{R}_{sf} + \bar{r}_f \tag{A.2}$$

where the vector $\bar{R}_{sf}$ is given by

$$\bar{R}_{sf} = \begin{bmatrix} 0 \\ R_e \text{ Cos } E \\ R_e \text{ Sin } E \end{bmatrix} \tag{A.3}$$

and E is the elevation angle of the radar boresight with respect to the local horizon plane.

The earth rotates about the polar axis with a constant angular velocity ω_e. The components of the earth's angular velocity vector in the RFCC system is given by

$$\bar{\omega}_f = \begin{bmatrix} \omega_{X_f} \\ \omega_{Y_f} \\ \omega_{Z_f} \end{bmatrix} = \begin{bmatrix} \omega_e \text{ Sin A Cos } \phi \\ \omega_e(\text{Cos E Sin } \phi - \text{Sin E Cos A Cos } \phi) \\ \omega_e(\text{Cos A Cos E Cos } \phi + \text{Sin E Sin } \phi) \end{bmatrix} \tag{A.4}$$

where A is the azimuth of the radar boresight with respect to the radar centered horizon coordinate system and ϕ is the geocentric latitude of the radar site.

1.2 Trajectory Integration

The trajectory integration involved in the prediction of the state vector will be performed by a second-order Taylor's series expansion in $\Delta t_n = t_{n+1} - t_n$ for the position vector $\bar{r}_f$ which gives

$$\bar{r}_f(t_{n+1}) = \bar{r}_f(t_n) + \dot{\bar{r}}_f(t_n)\Delta t_n + 1/2\ \ddot{\bar{r}}_f(t_n)\Delta t_n^2 \tag{A.5}$$

and consequently, the velocity vector is given by

$$\dot{\bar{r}}_f(t_{n+1}) = \dot{\bar{r}}_f(t_n) + \ddot{\bar{r}}_f(t_n)\Delta t_n \tag{A.6}$$

and

$$\lambda(t_{n+1}) = \lambda(t_n) \tag{A.7}$$

where $\ddot{\bar{r}}_f(t_n)$ is readily evaluated from Eq. (1.1) by using the current estimate of the state vector at t_n.

APPENDIX B

PROPAGATION OF ERRORS

Ignoring the process noise, the propagation of the state error covariance matrix from cycle to cycle is computed by

$$P(n+1/n) = \Phi(n+1,n)\ P(n)\ \Phi^{T}(n+1,n) \qquad (B.1)$$

in which $\Phi(n+1,n)$ is the state transition matrix expressed in terms of the appropriate filter coordinate system (RVCC).

The derivation of the Φ matrix starts with the first order Taylor's expansion of a set of nonlinear differential equations describing the target motion about some nominal solution of the state. For this specific development, the target motion Eq. (A.1) is approximated by

$$\ddot{\bar{r}}_f = -\frac{\rho g \lambda}{2} |\dot{\bar{r}}_f| \dot{\bar{r}}_f \qquad (B.2)$$

that is, all other accelerating forces except that due to the atmospheric drag are ignored.

The decoupled in-plane and out-of-plane transition matrices Φ_p and Φ_n in the RVCC system are given by

$$\Phi_p(n+1,n) = \begin{bmatrix} 1 & 0 & \Delta_n & 0 & 0 \\ 0 & 1 & 0 & \Delta_n & 0 \\ 0 & 0 & 1 & 0 & -D\dot{Z}_1\Delta_n \\ 0 & 0 & 0 & 1 & -D\dot{Z}_2\Delta n \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (B.3)$$

$$\Phi_n(n+1,n) = \begin{bmatrix} 1 & \Delta_n \\ 0 & 1 \end{bmatrix} \qquad (B.4)$$

APPENDIX B

RADAR/DPS INTERFACE SPECIFICATION

1.0 SCOPE

This specification defines the physical and functional interface between the Radar and the Data Processing Subsystem (DPS) of the Track Loop System (TLS). The TLS itself is a testbed derived from a Preliminary Ballistic Missile Defense System (PBMDS), which is in turn a representative but unreal environment for these studies. TLS is intended to have development and test capability, although realization of that capacity in actual testing is not now contemplated.

This specification corresponds to one which would be available prior to preparation of a PPR. To that end, its contents have been extracted from TDP documentation. Reference is made to that material as the source from which data here labelled "TBS" and "TBD" will be derived. In this publication "TBS" is used to identify material expected to be required during early stages of PPR preparation, while "TBD" denotes material which may be supplied later in the specification process. Material presented in italics, such as this paragraph, is illustrative or informative in nature, and would not normally appear in such a specification.

*Some of the paragraphs in this Interface Specification place constraints or requirements on the DPS; these have been identified by a * in front of that paragraph.*

2.0 APPLICABLE DOCUMENTS

2.1 TLS SYSTEM REQUIREMENTS 27332-6921-011, PART I, SECTION 1.1

2.2 TLS DATA PROCESSING SUBSYSTEM PERFORMANCE REQUIREMENTS (DPSPR) 27332-6921-011, PART II

2.3 TLS RADAR PERFORMANCE SPECIFICATION

2.4 TLS ENVIRONMENT AND THREAT MODEL DEFINITION

3.0 INTERFACE REQUIREMENTS

3.1 PHYSICAL INTERFACE

TBD

The physical interface is highly dependent on hardware selection for both the DPS and the Radar; in general, it will be irrelevant to the specification process through the Preliminary Design Review. Some portions of the physical interface may be specified during PPR, notably those relating to timing of signals and clock protocol.

3.2 FUNCTIONAL INTERFACE

The functional interface between the DP and the Radar shall consist of:

- Commands issued by the DP to the Radar
- Returns issued by the Radar to the DP
- Engagement Clock issued by the Radar to the DP

3.2.1 Radar Command Generation

a. The Radar will execute only the commands issued by the DP.

* b. Each command shall contain transmit, receive and synchronization data as described herein.

c. The radar shall transmit one pulse for each command which satisfies the radar and interface constraints. *Within PBMDS, but external to TLS, there are exceptions for pulse pairs and for verify pulses.*

* d. The DPS shall command a single receive window for each pulse. Within each receive window, the DPS shall command at least one receive gate.

3.2.2 Command Ordering

a. Radar shall execute commands in the order received.

* b. The DPS shall provide End of Transmission at least 1 msec before the scheduled execution time of the first command in the message.

3.2.3 Command Unpacking and Decoding

* The DPS shall issue commands in accordance with the format of TBD.

3.2.4 Timing Control

* a. The commanded times for Radar actions or for returns shall be in absolute time measured from a clock with nominal 1.68 second rollover. The least significant time bit shall be 6.25 nsec.

b. Radar shall determine all intermediate times needed to comply with command data for transmission and reception.

* c. The DPS shall not command receive windows which overlap. The receive window duration in each case shall be at least the uncompressed pulse length plus the desired range coverage.

* d. The DPS shall not command conflicting transmissions. Consecutive commands shall be separated in execution time by at least the uncompressed pulse length of the earlier plus TBS.

3.2.5 Command Contents

* a. The DPS shall provide a waveform identifier corresponding to one of the waveforms of Table A.2 (TBD) in each command.

* b. The DPS shall provide in each command both transmit and receive codes corresponding to direction cosine phase tapers.

* c. The DPS shall provide a receiver gain setting in each receive window.

* d. The DPS shall provide a signal processing mode code in each receive gate.

* e. The DPS shall provide a fixed signal acceptance threshold and threshold type selection for each receive gate.

* f. The DPS shall provide the range gate mark generation technique for each receive gate.

* g. The DPS shall provide the transmit power level for each command.

3.2.6 Return Contents

a. Radar shall return identifier data provided in the command.

b. Radar shall return actual range mark times and the signal amplitude at each range mark.

c. Radar shall return video signal amplitudes at commanded points. Amplitude shall be corrected by the Radar for stored instrumental errors.

d. For appropriate commanded signal-processing modes and waveforms, the Radar shall return direction cosines of the echo. Directional data shall be corrected by the Radar for stored instrumental errors.

e. For appropriate commands, the Radar shall return TBS wake array data.

f. Radar shall return noise level relevant to each amplitude in a return.

3.2.7 Error Handling

a. Radar shall return an error message for each discernible command not implemented. That message shall include a code corresponding to the reason for the failure of transmission. Among the reasons may be preemption, receive or transmit window overlap, insufficient time for transmission, and faulty command (internal inconsistency).

* b. The DPS shall initiate a record on permanent file of each error return.

* c. The DPS shall determine whether the fault is persistent or unique; if persistent, whether it is safety-related. (Definitions TBS). A persistent, safety-related fault shall cause test termination to be commanded by the DPS within TBS milliseconds; in particular, no command shall be issued for transmission by the Radar more than TBD milliseconds after a persistent, safety-related fault is detectable from returns.

3.2.8 Mode Change

* The DPS shall control all Radar mode changes through issuance of appropriate commands. The changes shall include startup and shutdown. Timeline constraints on mode changes and on preparation time for transmission are TBS.

3.2.9 Timing

a. Radar shall provide a master timing reference to DPS via TBS interface.

b. The clock shall provide 28 bits of data with a least significant bit of 6.25 nsec.

c. The resetting of the clock shall be entirely under Radar control. In consequence, its absolute value shall be regarded by the DPS as arbitrary. The Radar shall reset the clock within TBS milliseconds of startup, and shall not again reset it during the engagement.

3.2.10 Miscellaneous Requirements

a. Negative numbers crossing the interface shall be represented in two's complement form.

b. Radar coordinates shall be defined in accordance with Figure 3-1.

3.3 DATA REQUIREMENTS AND FORMATS

TBS

Data requirements and formats have been defined for TDP, and that definition might be carried over to the present document. However, it is not characteristic of systems such as TLS that details of bit positions, message formats, etc., would be known at this stage of development. However, the dynamic range, units, and least significant bit information is necessary in order to write performance requirements on radar command generation. The formats identification can be postponed until process design time.

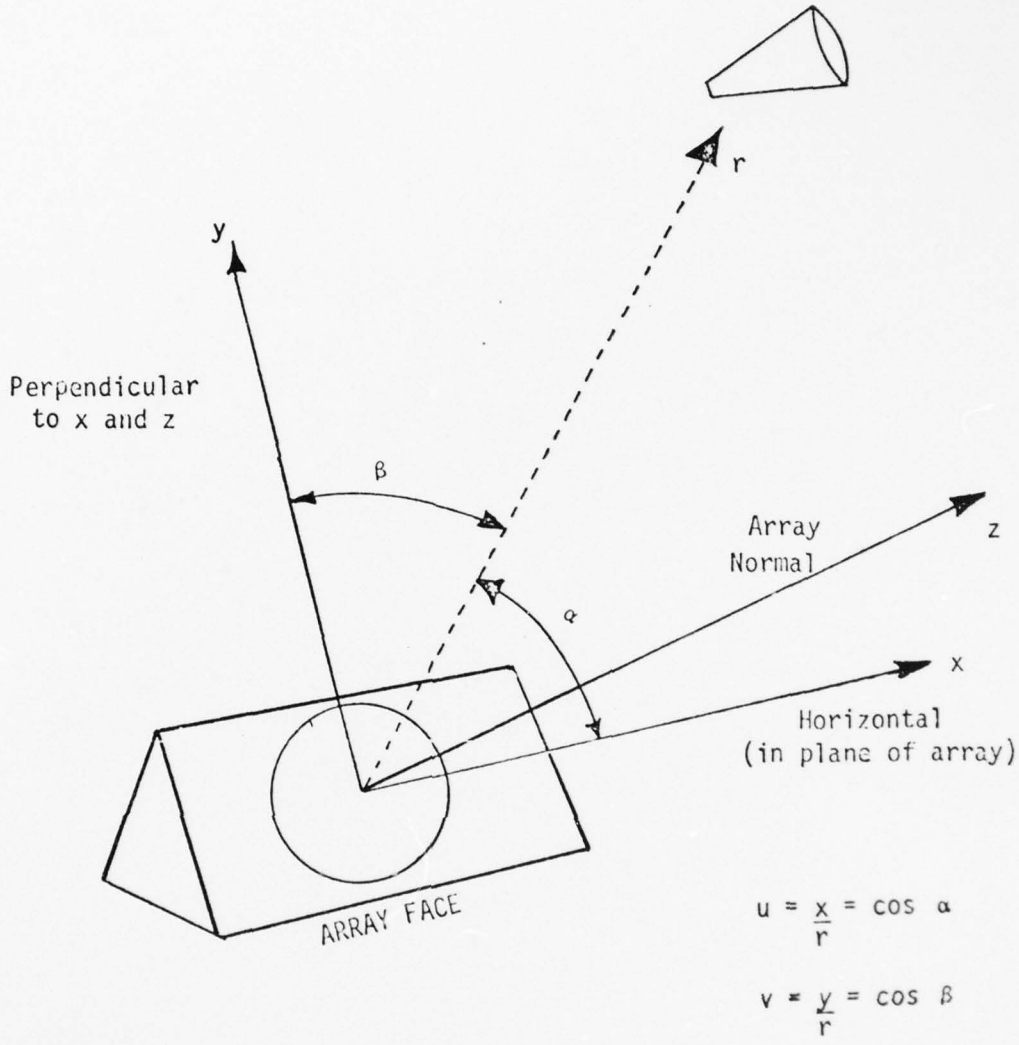
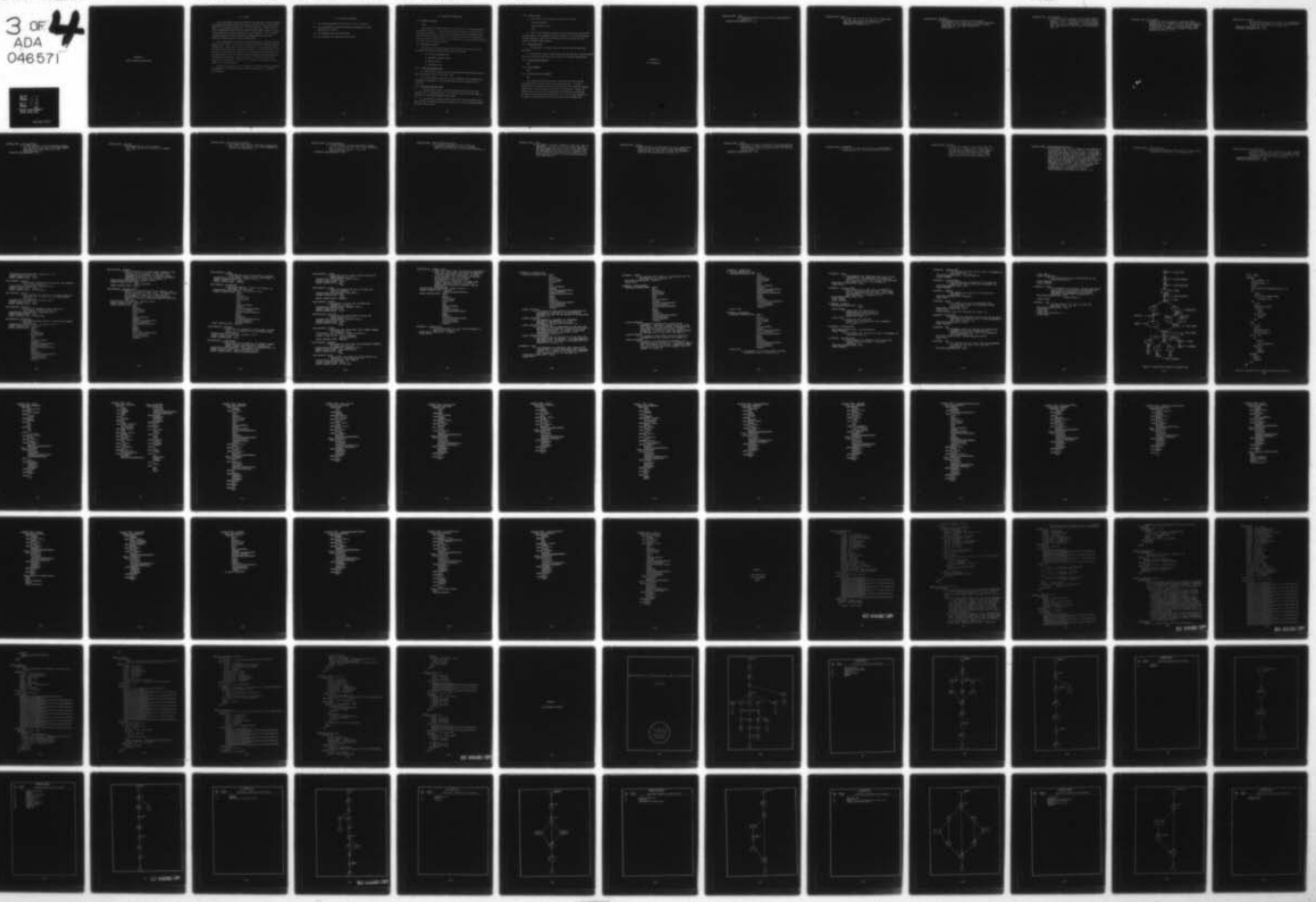


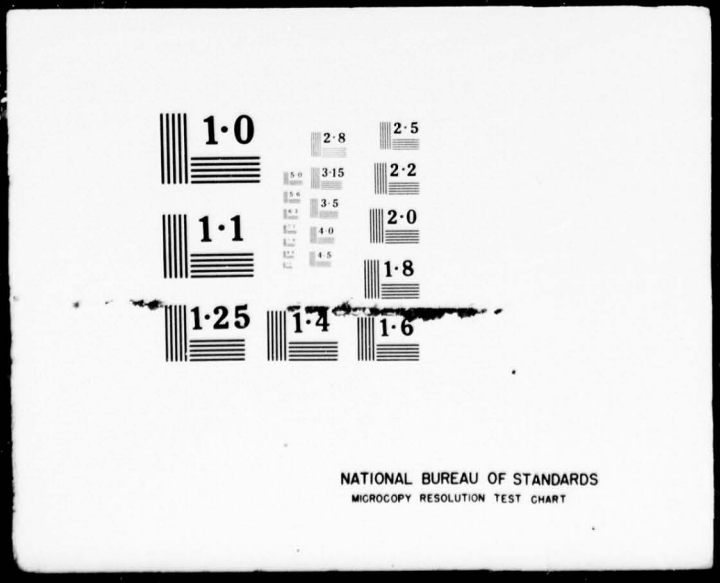
Figure 3-1. Radar Coordinate System

AD-A046 571 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY. SREP FINAL REPOR--ETC(U)
AUG 77 M W ALFORD, P H BROWNE, I F BURNS DASG60-75-C-0022
UNCLASSIFIED TRW-27332-6921-026-VOL-1 NL

3 OF 4
ADA
046571



3 OF 4
ADA
046571



ELEMENT TYPE A - INPUT INTERFACE

APPENDIX C

C^2/DPS INTERFACE SPECIFICATION

1.0 SCOPE

This specification defines the physical and functional interface between the Command and Communications (C^2) System and the Data Processing Subsystem (DPS) of the Track Loop System (TLS). The TLS itself is a testbed derived from a Preliminary Ballistic Missile Defense System (PBMDS), which is in turn a representative but unreal environment for these studies. TLS is intended to have development and test capability, although realization of that capacity in actual testing is not now contemplated.

This specification corresponds to one which would be available prior to preparation of a PPR. To that end, its contents have been extracted from TDP documentation. Reference is made to that material as the source from which data here labeled "TBS" and "TBD" will be derived. In this publication, "TBS" is used to identify material expected to be required during early stages of PPR preparation, while "TBD" denotes material which may be supplied later in the specification process. Material presented in italics, such as this paragraph, is illustrative or informative in nature, and would not normally appear in such a specification.

*Some of the paragraphs in this Interface Specification place constraints or requirements on the DPS; these have been identified by a * in front of that paragraph.*

2.0 APPLICABLE DOCUMENTS

2.1 TLS SYSTEM REQUIREMENTS 27332-6921-011, PART I, SECTION 1.1

2.2 TLS DATA PROCESSING SUBSYSTEM PERFORMANCE REQUIREMENTS (DPSPR) 27332-6921-011, PART II

2.3 TLS RADAR/DPS INTERFACE SPECIFICATION

2.4 TLS ENVIRONMENT AND THREAT MODEL DEFINITION

3.0 INTERFACE REQUIREMENTS

3.1 PHYSICAL INTERFACE

TBD

The physical interface between the C^2 and the DPS is entirely dependent on hardware selection. It will define polarity conventions, signal levels, and related parameters of interest only following PDR, and accountable only from the time of hardware integration. Although this section is required in such an interface specification, it will normally remain undetermined throughout the early stages of software design.

3.2 FUNCTIONAL INTERFACE

The functional interface between the C^2 and the DPS shall consist of messages of four types transmitted from the C^2 to the DPS.

- Initiate Engagement Mode
- Terminate Engagement Mode
- Handover Image
- Drop Image Track

3.2.1 Initiate Engagement Mode

* a. The DPS shall accept an Initiate Engagement Mode message from any of the following prior modes of the DPS: TBD.

* b. The DPS shall be prepared to accept a Handover Image message within TBS seconds of appearance of the Initiate Engagement Mode message at the interface.

3.2.2 Terminate Engagement Mode

* a. The DPS shall accept a Terminate Engagement Mode message at any time when it is in Engagement Mode. The DPS shall transition to TBD mode in response to a Terminate Engagement Mode message.

* b. The DPS shall command no Radar transmission with an execution time more than TBS milliseconds following appearance of a Terminate Engagement Mode message at the interface.

3.2.3 Handover Image

a. The contents of the Handover Image message shall be

Image designation

Image estimated state

TBS

b. The C^2 shall transmit no Handover Image message except when the DPS has been placed in the Engagement Mode by transmission of an Initiate Engagement Mode message at least TBS milliseconds earlier, and since the last Terminate Engagement Mode message.

3.2.4 Drop Image Track

a. The contents of the Drop Image Track message shall be the image designator.

b. The C^2 shall transmit no Drop Image Track message for an image designator unless that designator was previously included in a Handover Image message.

3.2.5 Message Acknowledgement

TBS

3.2.6 Error Handling

TBS

3.3 DATA REQUIREMENTS AND FORMATS

TBD

Data requirements and formats have been defined for TDP, and that definition might be carried over to the present document. It is not characteristic of systems such as TLS that details of bit positions, message formats, etc., would be known at this stage of development. However, the dynamic range, units, and least significant bit information is necessary in order to write performance requirements on the radar command generation. The formats identification can be postponed until process design time.

APPENDIX D

RSL TERMINOLOGY

```
ELEMENT_TYPE:  ALPHA
                  (* A PROCESSING STEP IN THE FUNCTIONAL REQUIREMENTS
                     DOMAIN, *),
   STRUCTURE APPLICABILITY:  NET,
```

```
ELEMENT_TYPE:  DATA
               (* A SINGLE ITEM OR SET OF DATA THAT IS SPECIFIED
                  AND THAT WILL EITHER BE REQUIRED IN THE
                  REAL-TIME SOFTWARE OR IS NEEDED FOR
                  DESCRIPTIVE PURPOSES. *),
```

```
ELEMENT_TYPE:  DECISION
               (* THE DECISION THAT HAS BEEN MADE TO ENABLE
                  REQUIREMENTS TO BE TAKEN FROM THE DPSPR TO THE PPR.
                  THIS MEANS THAT THE REQUIREMENTS ARE NOT SIMPLY
                  ALLOCATED, BUT HAVE BEEN SUBJECTED TO
                  DERIVATION. *).
```

```
ELEMENT_TYPE:  ENTITY_CLASS
               (* A GENERAL CLASS OF "OBJECTS" IN THE REAL WORLD
                  OUTSIDE THE DATA PROCESSING SYSTEM AND WHICH IS
                  IMPORTANT TO IT.  FOR EXAMPLE, AN ENTITY_CLASS
                  MIGHT BE RVS OR INTERCEPTORS.  THE ENTITY_TYPES
                  MIGHT BE DETECTION, POTENTIAL_RV, IDENTIFIED_RV,
                  ETC. *).
```

```
ELEMENT_TYPE:  ENTITY_TYPE
               (* A SPECIFIC TYPE OF "OBJECT" IN THE REAL WORLD
                  OUTSIDE THE DATA PROCESSING SYSTEM AND WHICH IS OF
                  IMPORTANCE TO IT.  WHEN A SPECIFIC TYPE OF "OBJECT"
                  IS DETERMINED TO EXIST IN AN ENTITY_CLASS, FILES
                  AND DATA MAY BE TEMPORARILY CREATED TO MAINTAIN
                  INFORMATION ABOUT IT. *),
```

```
ELEMENT_TYPE:  EVENT
                  (* AN IDENTIFIED POINT THAT EXISTS IN THE PROCESSING
                     OF ONE OR MORE R_NETS OR SUBNETS AND WHICH MAY
                     CAUSE THE ENABLEMENT OF AN R_NET. *).
   STRUCTURE APPLICABILITY:  NET.
   STRUCTURE APPLICABILITY:  PATH.
```

```
ELEMENT_TYPE: FILE
              (* AN AGGREGATION OF INSTANCES OF DATA, EACH INSTANCE
                 OF WHICH IS TREATED IN THE SAME MANNER. *).
```

```
ELEMENT_TYPE:  INPUT_INTERFACE
                (* A "PORT" BETWEEN THE DATA PROCESSING SYSTEM
                   AND THE REST OF THE BMD SYSTEM WHICH ACCEPTS
                   DATA FROM THE OTHER SYSTEM (E.G, THE
                   RADAR_RETURNS), *).
   STRUCTURE APPLICABILITY:  NET.
```

```
ELEMENT_TYPE: MESSAGE
              (* AN AGGREGATION OF DATA AND FILES
                 THAT PASS THROUGH AN INTERFACE AS A LOGICAL
                 UNIT. *).
```

```
ELEMENT_TYPE:  ORIGINATING_REQUIREMENT
               (* THE HIGHER LEVEL (DPSPR) REQUIREMENT FROM WHICH
                  LOWER LEVEL REQUIREMENTS (THE ONES DESCRIBED IN
                  THE RSL) ARE TRACEABLE, *),
```

```
ELEMENT_TYPE:  OUTPUT_INTERFACE
               (* A "PORT" BETWEEN THE DATA PROCESSING SYSTEM
                  AND THE REST OF THE BMD SYSTEM WHICH TRANSMITS
                  DATA TO THE OTHER SYSTEM  (E.G. THE
                  RADAR_COMMANDS). *).
  STRUCTURE APPLICABILITY:  NET.
```

```
ELEMENT_TYPE:  PERFORMANCE_REQUIREMENT
               (* AN ANALYTIC PERFORMANCE REQUIREMENT OR
                  NON-STIMULUS-RESPONSE TIMING REQUIREMENT
                  WHICH IS TO BE MET BY THE REAL-TIME SOFTWARE. *).
```

```
ELEMENT_TYPE: R_NET
              (* THE ORDER OF LOGICAL PROCESSING STEPS THAT MUST BE
                 PERFORMED.  AN R_NET MAY CONTAIN ANDS, ORS, AND
                 FOR EACH NODES; IT MUST BE ENABLED AND TERMINATED.
                 THE PROCESSING STEPS ARE ALPHAS OR SUBNETS WHICH
                 MAY BE EXPANDED INTO LOWER LEVELS OF DETAIL.  AN
                 R_NET MAY ALSO CONTAIN VALIDATION_POINTS, EVENTS,
                 AND INTERFACES. *).
```

```
ELEMENT_TYPE:  SOURCE
               (* SOURCE MATERIAL FOR REQUIREMENTS, I.E., ORIGINATING
                  POINT FOR ONE OR MORE ORIGINATING_REQUIREMENTS,
                  DOCUMENTATION OF TRADE-OFF STUDIES, OR BACKGROUND
                  MATERIAL FOR REQUIREMENTS ELEMENTS, *),
```

```
ELEMENT_TYPE:  SUBNET
               (* THE ORDER OF LOGICAL PROCESSING STEPS THAT MUST BE
                  PERFORMED IN ORDER TO PERFORM THE REQUIREMENTS OF
                  THE PROCESSING STEP THAT REPRESENTS IT AT THE NEXT
                  HIGHER LEVEL. *).
  STRUCTURE APPLICABILITY:  NET.
```

```
ELEMENT_TYPE:  SUBSYSTEM
                (* A PART OF THE BMD SYSTEM (SUCH AS RADAR) WHICH
                   COMMUNICATES WITH THE DATA PROCESSING SYSTEM. *).
```

```
ELEMENT_TYPE:  SYNONYM
               (* A SYNONYM IS MERELY AN ALTERNATE NAME THAT
                  CAN BE USED IN PLACE OF THE PRIME NAME.  IT
                  IS USED AS AN ABBREVIATION IN MOST CASES,
                  BUT MAY BE USED FOR OTHER REASONS ALSO.
                  NOTE:  IN AN <ELEMENT TYPE LIST>,  "ALL"
                  ALWAYS IMPLIES "ALL EXCEPT SYNONYM".  *),
```

```
ELEMENT_TYPE: UNSTRUCTURED_REQUIREMENT
        (* A REQUIREMENT THAT MUST BE PASSED TO THE DESIGNER OF
           THE REAL-TIME CODE BUT THAT DOES NOT FIT INTO THE
           STRUCTURED FRAMEWORK PROVIDED BY RSL.  THIS ELEMENT
           MIGHT BE USED BECAUSE THE REQUIREMENT IN QUESTION IS
           TOO UNIQUE TO JUSTIFY DEFINITION OF A NEW TYPE OF
           ELEMENT, A NEW RELATIONSHIP, OR A NEW ATTRIBUTE.  IT
           ALSO MIGHT BE USED BECAUSE THE REQUIREMENTS ANALYST
           DOES NOT CARE TO DESIGN A NEW CONCEPT TO FIT THE
           REQUIREMENT, OR BECAUSE THE REQUIREMENT IS CLEARLY
           A DESIGN LIMITATION THAT SHOULD BE DESCRIBED
           IN ENGLISH TEXT.  (AN EXAMPLE OF THE LAST
           REASON MIGHT BE PRECLUSION OF USING A
           MULTIPROCESSOR WITH ASSOCIATIVE MEMORY.) *),
```

```
ELEMENT_TYPE: VALIDATION_PATH
              (* THE PATH OF PROCESSING OVER WHICH THE QUANTITATIVE
                 VALIDATION TESTING WILL BE PERFORMED. *).
```

```
ELEMENT_TYPE:  VALIDATION_POINT
               (* A LOGICAL POINT IN THE PROCESSING AT WHICH TIMING,
                  VALUE, OR PRESENCE DATA MUST BE OBTAINABLE IN THE
                  REAL-TIME SOFTWARE IN ORDER TO VALIDATE THAT THE
                  REQUIREMENTS HAVE BEEN FULFILLED. *).
   STRUCTURE APPLICABILITY:  NET.
   STRUCTURE APPLICABILITY:  PATH.
```

```
ELEMENT_TYPE:  VERSION
               (* THE AGGREGATION OF REQUIREMENTS THAT ARE TO
                  APPLY AS A UNIT TO THE DATA PROCESSING
                  SYSTEM AT A PARTICULAR TIME.  LOOP_1,
                  LOOP_2, ETC., ARE VERSIONS, AS IS AN IOC
                  SYSTEM. *),

RELATIONSHIP:  ASSOCIATES
               (* TELLS WHICH DATA AND FILES
                  COME INTO EXISTENCE WHEN THE DATA
                  PROCESSING SYSTEM (SOME ALPHA) CREATES AN
                  INSTANCE OF AN ENTITY_CLASS OR AN R_NET IS
                  ENABLED. *),
   COMPLEMENTARY RELATIONSHIP:  ASSOCIATED ("WITH"),
   SUBJECT ELEMENT_TYPE:  ENTITY_CLASS
                          ENTITY_TYPE,
   OBJECT ELEMENT_TYPE:  DATA
                         FILE,

RELATIONSHIP:  COMPOSES
               (* TELLS WHICH ENTITY_TYPES ARE MEMBERS OF AN
                  ENTITY_CLASS. *),
   COMPLEMENTARY RELATIONSHIP:  COMPOSED ("OF"),
   SUBJECT ELEMENT_TYPE:  ENTITY_TYPE,
   OBJECT ELEMENT_TYPE:  ENTITY_CLASS,

RELATIONSHIP:  CONNECTS ("TO")
               (* TELLS WHICH SUBSYSTEM THE INPUT_INTERFACE OR
                  OUTPUT_INTERFACE COMMUNICATES WITH. *),
   COMPLEMENTARY RELATIONSHIP:  CONNECTED ("TO"),
   SUBJECT ELEMENT_TYPE:  INPUT_INTERFACE
                          OUTPUT_INTERFACE,
   OBJECT ELEMENT_TYPE:  SUBSYSTEM,

RELATIONSHIP:  CONSTRAINS
               (* IDENTIFIES TO WHICH VALIDATION_PATH(S) THE
                  PERFORMANCE_REQUIREMENT APPLIES. *),
   COMPLEMENTARY RELATIONSHIP:  CONSTRAINED ("BY"),
   SUBJECT ELEMENT_TYPE:  PERFORMANCE_REQUIREMENT,
   OBJECT ELEMENT_TYPE:  VALIDATION_PATH,

RELATIONSHIP:  CONTAINS
               (* TELLS THE IDENTITY OF EACH CONSTITUENT PART OF EACH
                  INSTANCE IN A FILE.  A DIRECT IMPLEMENTATION IN
                  SOFTWARE WOULD USE THIS RELATIONSHIP TO GIVE THE
                  MAKE-UP OF RECORDS IN FILES. *),
```

```
   COMPLEMENTARY RELATIONSHIP:  CONTAINED ("IN"),
   SUBJECT ELEMENT_TYPE:  FILE,
   OBJECT ELEMENT_TYPE:  DATA.

RELATIONSHIP:  CREATES
               (* TELLS WHICH PROCESSING STEPS CREATE THE INSTANCE
                  OF AN ENTITY_CLASS. *),
   COMPLEMENTARY RELATIONSHIP:  CREATED ("BY"),
   SUBJECT ELEMENT_TYPE:  ALPHA,
   OBJECT ELEMENT_TYPE:  ENTITY_CLASS.

RELATIONSHIP:  DELAYS
               (* THE ENABLEMENT OF R_NETS BY THE OBJECT EVENT IS
                  POSTPONED FOR THE AMOUNT OF TIME SPECIFIED BY
                  THE DATA. *),
   COMPLEMENTARY RELATIONSHIP:  DELAYED ("BY"),
   SUBJECT ELEMENT_TYPE:  DATA,
   OBJECT ELEMENT_TYPE:  EVENT.

RELATIONSHIP:  DESTROYS
               (* TELLS WHICH PROCESSING STEPS DESTROY AN
                  INSTANCE OF THE ENTITY_CLASS. *),
   COMPLEMENTARY RELATIONSHIP:  DESTROYED ("BY"),
   SUBJECT ELEMENT_TYPE:  ALPHA,
   OBJECT ELEMENT_TYPE:  ENTITY_CLASS.

RELATIONSHIP:  DOCUMENTS
               (* THE SUBJECT SOURCE MATERIAL DOCUMENTS THE OBJECT
                  ELEMENTS. *),
   COMPLEMENTARY RELATIONSHIP:  DOCUMENTED ("BY"),
   SUBJECT ELEMENT_TYPE:  SOURCE,
   OBJECT ELEMENT_TYPE:  ALPHA
                         DATA
                         DECISION
                         ENTITY_CLASS
                         ENTITY_TYPE
                         EVENT
                         FILE
                         INPUT_INTERFACE
                         MESSAGE
                         ORIGINATING_REQUIREMENT
                         OUTPUT_INTERFACE
                         PERFORMANCE_REQUIREMENT
                         R_NET
                         SUBNET
                         SUBSYSTEM
                         UNSTRUCTURED_REQUIREMENT
                         VALIDATION_PATH
                         VALIDATION_POINT
                         VERSION.
```

```
RELATIONSHIP:  ENABLES
               (* WHEN THE EVENT(S) IS (ARE) PASSED THROUGH BY THE
                  CONTROL FLOW ON AN R_NET, OR WHEN DATA ARE
                  AVAILABLE AT AN INTERFACE (AS DEFINED IN THE
                  INTERFACE DEFINITION), THE FUNCTIONAL PROCESSING
                  INDICATED ON THE R_NET CAN BE BEGUN, *),
   COMPLEMENTARY RELATIONSHIP:  ENABLED ("BY"),
   SUBJECT ELEMENT_TYPE:  EVENT
                          INPUT_INTERFACE,
   OBJECT ELEMENT_TYPE:  R_NET,

RELATIONSHIP:  EQUATES ("TO")
               (* DEFINES AN ALTERNATE NAME FOR AN ELEMENT.  THE
                  OBJECT IS CALLED THE PRIME NAME.  THE SUBJECT NAME
                  CAN BE USED FOR INPUT TO THE ASSM, BUT ALL
                  RELATIONSHIPS, ATTRIBUTES, AND STRUCTURES AS
                  DEFINED ARE ACTUALLY CHARACTERISTICS OF THE PRIME
                  NAME. *),
   COMPLEMENTARY RELATIONSHIP:  EQUATED ("TO"),
   SUBJECT ELEMENT_TYPE:  SYNONYM,
   OBJECT ELEMENT_TYPE:  ALPHA
                         DATA
                         DECISION
                         ENTITY_CLASS
                         ENTITY_TYPE
                         EVENT
                         FILE
                         INPUT_INTERFACE
                         MESSAGE
                         ORIGINATING_REQUIREMENT
                         OUTPUT_INTERFACE
                         PERFORMANCE_REQUIREMENT
                         R_NET
                         SOURCE
                         SUBNET
                         SUBSYSTEM
                         UNSTRUCTURED_REQUIREMENT
                         VALIDATION_PATH
                         VALIDATION_POINT
                         VERSION.
```

```
RELATIONSHIP:  FORMS
               (* INDICATES THE ALPHA WHICH DEFINES A MESSAGE
                  TO BE PASSED THROUGH AN OUTPUT_INTERFACE. *).
   COMPLEMENTARY RELATIONSHIP:  FORMED ("BY"),
   SUBJECT ELEMENT_TYPE:  ALPHA,
   OBJECT ELEMENT_TYPE:  MESSAGE.

RELATIONSHIP:  IMPLEMENTS
               (* TELLS THE VERSIONS TO WHICH THE ELEMENT, AS
                  DESCRIBED, APPLIES. *),
   COMPLEMENTARY RELATIONSHIP:  IMPLEMENTED ("BY"),
   SUBJECT ELEMENT_TYPE:  ALPHA
                          DATA
                          DECISION
                          ENTITY_CLASS
                          ENTITY_TYPE
                          EVENT
                          FILE
                          INPUT_INTERFACE
                          MESSAGE
                          ORIGINATING_REQUIREMENT
                          OUTPUT_INTERFACE
                          PERFORMANCE_REQUIREMENT
                          R_NET
                          SUBNET
                          SUBSYSTEM
                          UNSTRUCTURED_REQUIREMENT
                          VALIDATION_PATH
                          VALIDATION_POINT,
   OBJECT ELEMENT_TYPE:  VERSION.

RELATIONSHIP:  INCLUDES
               (* INDICATES THE HIERARCHICAL RELATIONSHIP BETWEEN
                  DATA.  IF A INCLUDES B, THEN OBTAINING A WILL
                  OBTAIN B. *),
   COMPLEMENTARY RELATIONSHIP:  INCLUDED ("IN"),
   SUBJECT ELEMENT_TYPE:  DATA,
   OBJECT ELEMENT_TYPE:  DATA.

RELATIONSHIP:  INCORPORATES
               (* INDICATES THAT THE SCOPE OF THE SUBJECT (HIGHER
                  LEVEL) ORIGINATING_REQUIREMENT INCLUDES THE
                  OBJECT (LOWER LEVEL) ORIGINATING_REQUIREMENT. *).
   COMPLEMENTARY RELATIONSHIP:  INCORPORATED ("IN"),
   SUBJECT ELEMENT_TYPE:  ORIGINATING_REQUIREMENT,
   OBJECT ELEMENT_TYPE:  ORIGINATING_REQUIREMENT.
```

```
RELATIONSHIP:  INPUTS
               (* INDICATES THAT THE NAMED ELEMENT INPUTS THE
                  OBJECT ELEMENT(S). *),
   COMPLEMENTARY RELATIONSHIP:  INPUT ("TO"),
   SUBJECT ELEMENT_TYPE:  ALPHA,
   OBJECT ELEMENT_TYPE:  DATA
                         FILE.

RELATIONSHIP:  MAKES
               (* TELLS THE IDENTITY OF DATA AND FILES THAT
                  MAKE UP A MESSAGE. *),
   COMPLEMENTARY RELATIONSHIP:  MADE ("BY"),
   SUBJECT ELEMENT_TYPE:  DATA
                          FILE.
   OBJECT ELEMENT_TYPE:  MESSAGE.

RELATIONSHIP:  ORDERS
               (* THE ELEMENT ON WHICH THE INSTANCES ARE
                  ORDERED IN A FILE. *),
   COMPLEMENTARY RELATIONSHIP:  ORDERED ("BY"),
   SUBJECT ELEMENT_TYPE:  DATA,
   OBJECT ELEMENT_TYPE:  FILE.

RELATIONSHIP:  OUTPUTS
               (* INDICATES THE NAMED ELEMENT OUTPUTS THE
                  OBJECT ELEMENT(S). *),
   COMPLEMENTARY RELATIONSHIP:  OUTPUT ("FROM"),
   SUBJECT ELEMENT_TYPE:  ALPHA,
   OBJECT ELEMENT_TYPE:  DATA
                         FILE.

RELATIONSHIP:  PASSES
               (* INDICATES THE INFORMATION WHICH PASSES THROUGH
                  THE INTERFACE. *),
   COMPLEMENTARY RELATIONSHIP:  PASSED ("THROUGH"),
   SUBJECT ELEMENT_TYPE:  INPUT_INTERFACE
                          OUTPUT_INTERFACE.
   OBJECT ELEMENT_TYPE:  MESSAGE.

RELATIONSHIP:  RECORDS
               (* INDICATES THAT THE NAMED VALIDATION_POINT RECORDS
                  THE OBJECT ELEMENTS. *),
   COMPLEMENTARY RELATIONSHIP:  RECORDED ("BY"),
   SUBJECT ELEMENT_TYPE:  VALIDATION_POINT,
   OBJECT ELEMENT_TYPE:  DATA
                         FILE.

RELATIONSHIP:  SETS
               (* TELLS WHICH ALPHA SETS THE ENTITY_TYPE OF AN
                  INSTANCE IN AN ENTITY_CLASS. *),
   COMPLEMENTARY RELATIONSHIP:  SET ("BY"),
   SUBJECT ELEMENT_TYPE:  ALPHA,
   OBJECT ELEMENT_TYPE:  ENTITY_TYPE.
```

```
RELATIONSHIP:  TRACES ("TO")
               (* TELLS THE HIGHER LEVEL (ORIGINATING) REQUIREMENT
                  FROM WHICH THE ELEMENT WAS TRACED (ALLOCATED).
                  A DEFAULT ORIGINATING REQUIREMENT IS "NONE",
                  WHICH INDICATES THAT THE ELEMENT IS DERIVED.
                  WE WOULD EXPECT THAT THE ATTRIBUTE
                  "ARTIFICIALITY" WOULD HAVE VALUE
                  "ARTIFICIAL" IF THE ELEMENT IS TRACED FROM
                  "NONE".  HOWEVER, THIS MAY BE UNTRUE IN
                  CASES WHERE AN ARBITRARY HEURISTIC MUST BE
                  EMPLOYED. *),
   COMPLEMENTARY RELATIONSHIP:  TRACED ("FROM"),
   SUBJECT ELEMENT_TYPE:  DECISION
                          ORIGINATING_REQUIREMENT,
   OBJECT ELEMENT_TYPE:  ALPHA
                         DATA
                         DECISION
                         ENTITY_CLASS
                         ENTITY_TYPE
                         EVENT
                         FILE
                         INPUT_INTERFACE
                         MESSAGE
                         OUTPUT_INTERFACE
                         PERFORMANCE_REQUIREMENT
                         R_NET
                         SUBNET
                         SUBSYSTEM
                         UNSTRUCTURED_REQUIREMENT
                         VALIDATION_PATH
                         VALIDATION_POINT
                         VERSION.

ATTRIBUTE:  ALTERNATIVES
            (* THE ALTERNATIVES THAT HAVE BEEN ENVISIONED TO
               RESOLVE THE PROBLEM. *),
   APPLICABLE ELEMENT_TYPE:  DECISION.
   VALUE: TEXT.
```

```
ATTRIBUTE: ARTIFICIALITY.
   APPLICABLE ELEMENT_TYPE:  ALPHA
                             DATA
                             DECISION
                             ENTITY_CLASS
                             ENTITY_TYPE
                             EVENT
                             FILE
                             INPUT_INTERFACE
                             MESSAGE
                             ORIGINATING_REQUIREMENT
                             OUTPUT_INTERFACE
                             PERFORMANCE_REQUIREMENT
                             R_NET
                             SOURCE
                             SUBNET
                             SUBSYSTEM
                             UNSTRUCTURED_REQUIREMENT
                             VALIDATION_PATH
                             VALIDATION_POINT
                             VERSION.
   VALUE: ARTIFICIAL
             (* THE ELEMENT HAS BEEN DEFINED FOR EXPLANATORY OR
                EXECUTABILITY PURPOSES IN THE REQUIREMENTS
                STATEMENT AND NEED NOT BE PRESENT IN THE REAL-TIME
                SOFTWARE. *).
   VALUE: VALIDATION
             (* THE ELEMENT IS NECESSARY FOR DESCRIBING
                PERFORMANCE REQUIREMENTS BUT IS NOT
                REQUIRED IN THE REAL-TIME SOFTWARE. *).
   VALUE: IMPLEMENT_APPROXIMATELY
             (* THE ELEMENT MUST BE IMPLEMENTED IN THE REAL-TIME
                SOFTWARE, BUT THE PRECISE IMPLEMENTATION IS LEFT T(
                THE PROCESS DESIGNER.  OF COURSE, THE DETAILED
                IMPLEMENTATION MUST BE VALIDATED BY THE
                REQUIREMENTS ANALYST. *).
   VALUE: IMPLEMENT_PRECISELY
             (* THE ELEMENT MUST BE IMPLEMENTED IN THE REAL-TIME
                SOFTWARE EXACTLY AS DEFINED;  NO CHANGES SHOULD BE
                CONSIDERED UNTIL PERMISSION IS OBTAINED FROM THE
                REQUIREMENTS ANALYST. *).

ATTRIBUTE:  BETA
             (* THIS PROVIDES THE PROCEDURAL CODE (WHICH IS NOT
                INTERPRETED BY THE RSL PROCESSORS) FOR FUNCTIONAL
                MODELS.  IT IS PASSED TO THE SIMULATION GENERATOR
                AND, SUBSEQUENTLY, TO THE COMPILER. *).
   APPLICABLE ELEMENT_TYPE:  ALPHA.
   VALUE: TEXT.
```

```
ATTRIBUTE:  CHOICE
              (* THE DECISION FROM AMONG THE ALTERNATIVES WITH THE
                 RATIONALE FOR THE DECISION. *),
   APPLICABLE ELEMENT_TYPE:  DECISION,
   VALUE: TEXT.

ATTRIBUTE:  COMPLETENESS,
   APPLICABLE ELEMENT_TYPE:  ALPHA
                             DATA
                             DECISION
                             ENTITY_CLASS
                             ENTITY_TYPE
                             EVENT
                             FILE
                             INPUT_INTERFACE
                             MESSAGE
                             ORIGINATING_REQUIREMENT
                             OUTPUT_INTERFACE
                             PERFORMANCE_REQUIREMENT
                             R_NET
                             SOURCE
                             SUBNET
                             SUBSYSTEM
                             UNSTRUCTURED_REQUIREMENT
                             VALIDATION_PATH
                             VALIDATION_POINT
                             VERSION,
   VALUE: INCOMPLETE
              (* THE ELEMENT'S DESCRIPTION IS KNOWN TO BE
                 INCOMPLETE.  THEREFORE, READERS SHOULD BE AWARE
                 THAT, EVEN IF ALL RELATIONSHIPS, ATTRIBUTES, AND
                 STRUCTURES ARE STATED, THE ELEMENT IS STILL
                 INCOMPLETE.  INFORMATION ABOUT THE ELEMENT SHOULD
                 BE EMPLOYED ONLY AT THE USER'S OWN RISK. *),
   VALUE: COMPLETE
              (* THE ELEMENT'S DESCRIPTION SHOULD BE ASSUMED TO
                 BE COMPLETE AND WILL PROBABLY NOT CHANGE. *),
   VALUE: CHANGEABLE
              (* ALTHOUGH ALL RELATIONSHIPS, ATTRIBUTES, AND
                 STRUCTURES MAY BE DEFINED FOR THE ELEMENT, SOME OF
                 THEM WILL PROBABLY BE CHANGED.  INFORMATION ABOUT
                 THE ELEMENT IS BELIEVED TO BE CORRECT, BUT IT IS
                 SUBJECT TO CHANGE. *),
```

```
ATTRIBUTE: DESCRIPTION
(* TEXTUAL DESCRIPTION, *),
   APPLICABLE ELEMENT_TYPE: ALPHA
                            DATA
                            DECISION
                            ENTITY_CLASS
                            ENTITY_TYPE
                            EVENT
                            FILE
                            INPUT_INTERFACE
                            MESSAGE
                            ORIGINATING_REQUIREMENT
                            OUTPUT_INTERFACE
                            PERFORMANCE_REQUIREMENT
                            R_NET
                            SOURCE
                            SUBNET
                            SUBSYSTEM
                            UNSTRUCTURED_REQUIREMENT
                            VALIDATION_PATH
                            VALIDATION_POINT
                            VERSION,

   VALUE: TEXT,

ATTRIBUTE: ENTERED_BY,
   APPLICABLE ELEMENT_TYPE: ALPHA
                            DATA
                            DECISION
                            ENTITY_CLASS
                            ENTITY_TYPE
                            EVENT
                            FILE
                            INPUT_INTERFACE
                            MESSAGE
                            ORIGINATING_REQUIREMENT
                            OUTPUT_INTERFACE
                            PERFORMANCE_REQUIREMENT
                            R_NET
                            SOURCE
                            SUBNET
                            SUBSYSTEM
                            UNSTRUCTURED_REQUIREMENT
                            VALIDATION_PATH
                            VALIDATION_POINT
                            VERSION.

   VALUE: TEXT
            (* THE IDENTITY OF THE LAST PERSON TO ENTER
               INFORMATION ABOUT THE ELEMENT. *),
```

```
ATTRIBUTE: GAMMA
              (* THIS PROVIDES THE PROCEDURAL CODE (WHICH IS NOT
                 INTERPRETED BY RSL PROCESSORS) FOR ANALYTIC
                 MODELS.  IT IS PASSED TO THE SIMULATION GENERATOR
                 AND, SUBSEQUENTLY, TO THE COMPILER. *).
   APPLICABLE ELEMENT_TYPE: ALPHA.
   VALUE: TEXT.

ATTRIBUTE: INITIAL_VALUE
              (* THE INITIAL_VALUE A DATA ITEM IS REQUIRED TO
                 HAVE IN THE IMPLEMENTED SOFTWARE.  THIS VALUE
                 WILL BE ASSUMED BY THE DATA ITEM WHEN IT COMES
                 INTO EXISTENCE IN A SIMULATION. *).
   APPLICABLE ELEMENT_TYPE: DATA.
   VALUE: NAMED.
   VALUE: NUMERIC.

ATTRIBUTE: LOCALITY.
   APPLICABLE ELEMENT_TYPE: DATA
                            FILE.
   VALUE: GLOBAL
              (* GLOBAL DATA AND FILES MAY BE
                 ASSOCIATED WITH ENTITY_TYPES OR
                 ENTITY-CLASSES OR MAY BE IN THE
                 GLOBAL DATA BASE. *).
   VALUE: LOCAL
              (* LOCAL DATA AND FILES ARE
                 CREATED AND INITIALIZED FOR EACH ENABLEMENT
                 OF AN R_NET. *).

ATTRIBUTE: MAXIMUM_TIME.
   APPLICABLE ELEMENT_TYPE: VALIDATION_PATH.
   VALUE: NUMERIC
              (* THE MAXIMUM TIME THAT CAN BE TAKEN TO TRAVERSE THE
                 VALIDATION PATH. *).

ATTRIBUTE: MAXIMUM_VALUE
              (* THE MAXIMUM_VALUE APPLIES TO DATA VALUES AND
                 EMPLOYS THE UNITS STATED IN THE UNITS
                 ATTRIBUTE. *).
   APPLICABLE ELEMENT_TYPE: DATA.
   VALUE: NUMERIC.
```

```
ATTRIBUTE:  MINIMUM_TIME
              (* THE MINIMUM TIME THAT CAN BE TAKEN TO TRAVERSE THE
                 VALIDATION PATH. *).
   APPLICABLE ELEMENT_TYPE:  VALIDATION_PATH.
   VALUE: NUMERIC.

ATTRIBUTE:  MINIMUM_VALUE
              (* THE MINIMUM_VALUE APPLIES TO DATA VALUES AND
                 EMPLOYS THE UNITS IN THE UNITS ATTRIBUTE. *).
   APPLICABLE ELEMENT_TYPE:  DATA.
   VALUE: NUMERIC.

ATTRIBUTE:  PROBLEM
              (* THE PROBLEM THAT HAS LED TO THE NEED FOR A
                 DECISION. *).
   APPLICABLE ELEMENT_TYPE:  DECISION.
   VALUE: TEXT.

ATTRIBUTE:  RANGE
              (* THE RANGE OF THE DATA IS ENUMERATED HERE.
                 IT IS MEANINGFUL ONLY IF ENUMERATION IS THE
                 VALUE OF TYPE. *).
   APPLICABLE ELEMENT_TYPE:  DATA.
   VALUE: TEXT
   (* THE ALLOWED VALUES ARE SEPARATED BY COMMAS. *).

ATTRIBUTE:  RESOLUTION
              (* DESCRIBES THE REQUIRED MAXIMUM VALUE OF THE LEAST
                 SIGNIFICANT BIT FOR THE DATA IN UNITS DESCRIBED IN
                 THE UNITS ATTRIBUTE. *).
   APPLICABLE ELEMENT_TYPE:  DATA.
   VALUE: NUMERIC.

ATTRIBUTE:  TEST
              (* PROCEDURAL CODE WHICH DEFINES THE COMPUTATIONS
                 NECESSARY TO TEST THE SATISFACTION OF A
                 PERFORMANCE_REQUIREMENT USING DATA RECORDED BY
                 VALIDATION_POINTS. *).
   APPLICABLE ELEMENT_TYPE:  PERFORMANCE_REQUIREMENT.
   VALUE: TEXT.

ATTRIBUTE:  TYPE
              (* THE TYPE FOR THE DATA ITEMS WHICH ARE REFERENCED
                 ON R_NETS OR ARE USED IN BETA OR GAMMA
                 SIMULATIONS. *).
   APPLICABLE ELEMENT_TYPE:  DATA.
```

```
   VALUE: REAL,
   VALUE: ENUMERATION
             (* THE ALLOWED VALUES MUST BE SPECIFIED IN THE
                RANGE ATTRIBUTE. *),
   VALUE: BOOLEAN,
   VALUE: INTEGER.

ATTRIBUTE:  UNITS
             (* THE CONFIGURATION MANAGEMENT MANAGER MUST CREATE
                THE ATTRIBUTE VALUES THAT MAY BE EMPLOYED IN
                DESCRIBING THE REQUIREMENTS FOR THE DATA
                PROCESSING SYSTEM UNDER CONSIDERATION. *),
   APPLICABLE ELEMENT_TYPE:  DATA
                             VALIDATION_PATH,
   VALUE: NAMED.

ATTRIBUTE:  USE
             (* THE APPLICABILITY OF TYPE AND RANGE ARE
                SPECIFIED IN THE USE. *),
   APPLICABLE ELEMENT_TYPE:  DATA,
   VALUE: BETA,
   VALUE: BOTH
   (* BOTH BETA AND GAMMA *),
   VALUE: GAMMA.
```

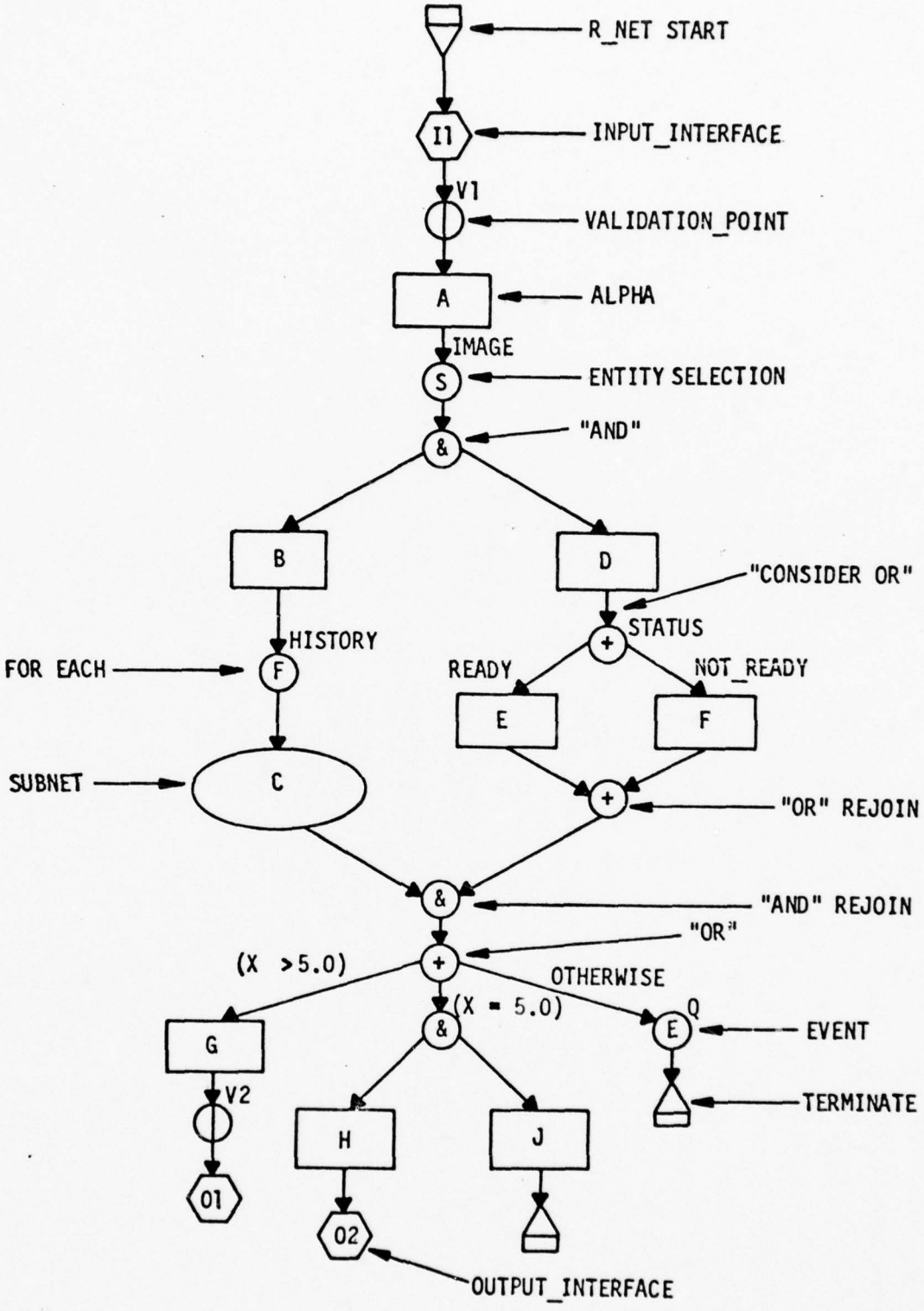


Figure D-1 Sample R_NET Structure in Graphical Form

```
R_NET:  SAMPLE.
STRUCTURE:
      INPUT_INTERFACE   I1
      VALIDATION_POINT   V1
      ALPHA A
      SELECT ENTITY_CLASS IMAGE SUCH THAT (Y < Z)
      DO
          ALPHA B
          FOR EACH FILE HISTORY RECORD
              DO SUBNET C END
      AND
          ALPHA D
          CONSIDER DATA STATUS
          IF (READY)
              ALPHA E
          OR (NOT_READY)
              ALPHA F
          END
      END
      IF (X > 5.0)
          ALPHA G
          VALIDATION_POINT V2
          OUTPUT_INTERFACE O1
      OR (X = 5.0)
          DO
              ALPHA H
              OUTPUT_INTERFACE O2
          AND ALPHA J
              ALPHA J
              TERMINATE
      OTHERWISE
          EVENT Q
          TERMINATE
      END
END.
```

Figure D-2 Equivalent RSL for Sample R_NET Structure (Figure D-1)

APPENDIX E

RSL SUMMARY BY ELEMENT-TYPE

```
ELEMENT_TYPE:  ALPHA
   LEGAL RELATIONSHIPS:
     CREATES:
          ENTITY_CLASS
     DESTROYS:
          ENTITY_CLASS
     FORMS:
          MESSAGE
     IMPLEMENTS:
          VERSION
     INPUTS:
          DATA
          FILE
     OUTPUTS:
          DATA
          FILE
     SETS:
          ENTITY_TYPE
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     BETA:
          TEXT
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
     GAMMA:
          TEXT
```

```
ELEMENT_TYPE:  DATA
   LEGAL RELATIONSHIPS:
      DELAYS:
            EVENT
      IMPLEMENTS:
            VERSION
      INCLUDES:
            DATA
      MAKES:
            MESSAGE
      ORDERS:
            FILE
      ASSOCIATED ("WITH"):
            ENTITY_CLASS
            ENTITY_TYPE
      CONTAINED ("IN"):
            FILE
      DOCUMENTED ("BY"):
            SOURCE
      EQUATED ("TO"):
            SYNONYM
      INCLUDED ("IN"):
            DATA
      INPUT ("TO"):
            ALPHA
      OUTPUT ("FROM"):
            ALPHA
      RECORDED ("BY"):
            VALIDATION_POINT
      TRACED ("FROM"):
            DECISION
            ORIGINATING_REQUIREMENT

LEGAL ATTRIBUTES:
   ARTIFICIALITY:
            ARTIFICIAL
            VALIDATION
            IMPLEMENT_APPROXIMATELY
            IMPLEMENT_PRECISELY
   COMPLETENESS:
            INCOMPLETE
            COMPLETE
            CHANGEABLE
   DESCRIPTION:
            TEXT
   ENTERED_BY:
            TEXT
   INITIAL_VALUE:
            NAMED
            NUMERIC
   LOCALITY:
            GLOBAL
            LOCAL
   MAXIMUM_VALUE:
            NUMERIC
   MINIMUM_VALUE:
            NUMERIC
   RANGE:
            TEXT
   RESOLUTION:
            NUMERIC
   TYPE:
            REAL
            ENUMERATION
            BOOLEAN
            INTEGER
   UNITS:
            NAMED
   USE:
            BETA
            BOTH
            GAMMA
```

```
ELEMENT_TYPE:  DECISION
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     TRACES ("TO"):
          ALPHA
          DATA
          DECISION
          ENTITY_CLASS
          ENTITY_TYPE
          EVENT
          FILE
          INPUT_INTERFACE
          MESSAGE
          OUTPUT_INTERFACE
          PERFORMANCE_REQUIREMENT
          R_NET
          SUBNET
          SUBSYSTEM
          UNSTRUCTURED_REQUIREMENT
          VALIDATION_PATH
          VALIDATION_POINT
          VERSION
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ALTERNATIVES:
          TEXT
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     CHOICE:
          TEXT
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
     PROBLEM:
          TEXT
```

```
ELEMENT_TYPE:  ENTITY_CLASS
   LEGAL RELATIONSHIPS:
      ASSOCIATES:
            DATA
            FILE
      IMPLEMENTS:
            VERSION
      COMPOSED ("OF"):
            ENTITY_TYPE
      CREATED ("BY"):
            ALPHA
      DESTROYED ("BY"):
            ALPHA
      DOCUMENTED ("BY"):
            SOURCE
      EQUATED ("TO"):
            SYNONYM
      TRACED ("FROM"):
            DECISION
            ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
      ARTIFICIALITY:
            ARTIFICIAL
            VALIDATION
            IMPLEMENT_APPROXIMATELY
            IMPLEMENT_PRECISELY
      COMPLETENESS:
            INCOMPLETE
            COMPLETE
            CHANGEABLE
      DESCRIPTION:
            TEXT
      ENTERED_BY:
            TEXT
```

```
ELEMENT_TYPE:  ENTITY_TYPE
   LEGAL RELATIONSHIPS:
     ASSOCIATES:
          DATA
          FILE
     COMPOSES:
          ENTITY_CLASS
     IMPLEMENTS:
          VERSION
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     SET ("BY"):
          ALPHA
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  EVENT
   LEGAL RELATIONSHIPS:
     ENABLES:
          R_NET
     IMPLEMENTS:
          VERSION
     DELAYED ("BY"):
          DATA
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  FILE
   LEGAL RELATIONSHIPS:
      CONTAINS:
            DATA
      IMPLEMENTS:
            VERSION
      MAKES:
            MESSAGE
      ASSOCIATED ("WITH"):
            ENTITY_CLASS
            ENTITY_TYPE
      DOCUMENTED ("BY"):
            SOURCE
      EQUATED ("TO"):
            SYNONYM
      INPUT ("TO"):
            ALPHA
      ORDERED ("BY"):
            DATA
      OUTPUT ("FROM"):
            ALPHA
      RECORDED ("BY"):
            VALIDATION_POINT
      TRACED ("FROM"):
            DECISION
            ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
      ARTIFICIALITY:
            ARTIFICIAL
            VALIDATION
            IMPLEMENT_APPROXIMATELY
            IMPLEMENT_PRECISELY
      COMPLETENESS:
            INCOMPLETE
            COMPLETE
            CHANGEABLE
      DESCRIPTION:
            TEXT
      ENTERED_BY:
            TEXT
      LOCALITY:
            GLOBAL
            LOCAL
```

```
ELEMENT_TYPE:  INPUT_INTERFACE
   LEGAL RELATIONSHIPS:
     CONNECTS ("TO"):
          SUBSYSTEM
     ENABLES:
          R_NET
     IMPLEMENTS:
          VERSION
     PASSES:
          MESSAGE
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  MESSAGE
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     FORMED ("BY"):
          ALPHA
     MADE ("BY"):
          DATA
          FILE
     PASSED ("THROUGH"):
          INPUT_INTERFACE
          OUTPUT_INTERFACE
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  ORIGINATING_REQUIREMENT
   LEGAL RELATIONSHIPS:
      IMPLEMENTS:
            VERSION
      INCORPORATES:
            ORIGINATING_REQUIREMENT
      TRACES ("TO"):
            ALPHA
            DATA
            DECISION
            ENTITY_CLASS
            ENTITY_TYPE
            EVENT
            FILE
            INPUT_INTERFACE
            MESSAGE
            OUTPUT_INTERFACE
            PERFORMANCE_REQUIREMENT
            R_NET
            SUBNET
            SUBSYSTEM
            UNSTRUCTURED_REQUIREMENT
            VALIDATION_PATH
            VALIDATION_POINT
            VERSION
      DOCUMENTED ("BY"):
            SOURCE
      EQUATED ("TO"):
            SYNONYM
      INCORPORATED ("IN"):
            ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
      ARTIFICIALITY:
            ARTIFICIAL
            VALIDATION
            IMPLEMENT_APPROXIMATELY
            IMPLEMENT_PRECISELY
      COMPLETENESS:
            INCOMPLETE
            COMPLETE
            CHANGEABLE
      DESCRIPTION:
            TEXT
      ENTERED_BY:
            TEXT
```

```
ELEMENT_TYPE:  OUTPUT_INTERFACE
   LEGAL RELATIONSHIPS:
     CONNECTS ("TO"):
          SUBSYSTEM
     IMPLEMENTS:
          VERSION
     PASSES:
          MESSAGE
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  PERFORMANCE_REQUIREMENT
   LEGAL RELATIONSHIPS:
     CONSTRAINS:
          VALIDATION_PATH
     IMPLEMENTS:
          VERSION
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
     TEST:
          TEXT
```

```
ELEMENT_TYPE:  R_NET
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     DOCUMENTED ("BY"):
          SOURCE
     ENABLED ("BY"):
          EVENT
          INPUT_INTERFACE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
   LEGAL STRUCTURE ELEMENT_TYPES:
     ALPHA
     EVENT
     INPUT_INTERFACE
     OUTPUT_INTERFACE
     SUBNET
     VALIDATION_POINT
```

```
ELEMENT_TYPE:  SOURCE
   LEGAL RELATIONSHIPS:
     DOCUMENTS:
          ALPHA
          DATA
          DECISION
          ENTITY_CLASS
          ENTITY_TYPE
          EVENT
          FILE
          INPUT_INTERFACE
          MESSAGE
          ORIGINATING_REQUIREMENT
          OUTPUT_INTERFACE
          PERFORMANCE_REQUIREMENT
          R_NET
          SUBNET
          SUBSYSTEM
          UNSTRUCTURED_REQUIREMENT
          VALIDATION_PATH
          VALIDATION_POINT
          VERSION
     EQUATED ("TO"):
          SYNONYM
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  SUBNET
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
   LEGAL STRUCTURE ELEMENT_TYPES:
     ALPHA
     EVENT
     OUTPUT_INTERFACE
     SUBNET
     VALIDATION_POINT
```

```
ELEMENT_TYPE:  SUBSYSTEM
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     CONNECTED ("TO"):
          INPUT_INTERFACE
          OUTPUT_INTERFACE
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  SYNONYM
   LEGAL RELATIONSHIPS:
     EQUATES ("TO"):
           ALPHA
           DATA
           DECISION
           ENTITY_CLASS
           ENTITY_TYPE
           EVENT
           FILE
           INPUT_INTERFACE
           MESSAGE
           ORIGINATING_REQUIREMENT
           OUTPUT_INTERFACE
           PERFORMANCE_REQUIREMENT
           R_NET
           SOURCE
           SUBNET
           SUBSYSTEM
           UNSTRUCTURED_REQUIREMENT
           VALIDATION_PATH
           VALIDATION_POINT
           VERSION
   NO LEGAL ATTRIBUTES
```

```
ELEMENT_TYPE:  UNSTRUCTURED_REQUIREMENT
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE:  VALIDATION_PATH
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     CONSTRAINED ("BY"):
          PERFORMANCE_REQUIREMENT
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
     MAXIMUM_TIME:
          NUMERIC
     MINIMUM_TIME:
          NUMERIC
     UNITS:
          NAMED
   LEGAL PATH ELEMENT_TYPES:
     EVENT
     VALIDATION_POINT
```

```
ELEMENT_TYPE:  VALIDATION_POINT
   LEGAL RELATIONSHIPS:
     IMPLEMENTS:
          VERSION
     RECORDS:
          DATA
          FILE
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

```
ELEMENT_TYPE: VERSION
   LEGAL RELATIONSHIPS:
     DOCUMENTED ("BY"):
          SOURCE
     EQUATED ("TO"):
          SYNONYM
     IMPLEMENTED ("BY"):
          ALPHA
          DATA
          DECISION
          ENTITY_CLASS
          ENTITY_TYPE
          EVENT
          FILE
          INPUT_INTERFACE
          MESSAGE
          ORIGINATING_REQUIREMENT
          OUTPUT_INTERFACE
          PERFORMANCE_REQUIREMENT
          R_NET
          SUBNET
          SUBSYSTEM
          UNSTRUCTURED_REQUIREMENT
          VALIDATION_PATH
          VALIDATION_POINT
     TRACED ("FROM"):
          DECISION
          ORIGINATING_REQUIREMENT
   LEGAL ATTRIBUTES:
     ARTIFICIALITY:
          ARTIFICIAL
          VALIDATION
          IMPLEMENT_APPROXIMATELY
          IMPLEMENT_PRECISELY
     COMPLETENESS:
          INCOMPLETE
          COMPLETE
          CHANGEABLE
     DESCRIPTION:
          TEXT
     ENTERED_BY:
          TEXT
```

APPENDIX F

TRACK LOOP SYSTEM
RSL REQUIREMENTS
KERNEL

```
R_NET:  CC_RESPONSE.
     REFERS TO:
          ALPHA:  ACKNOWLEDGE
          ALPHA:  CC_ERROR_PROCESSING
          ALPHA:  ENGAGEMENT_INITIATION
          ALPHA:  STARTER
          ALPHA:  TERM_ENGAGEMENT
          ALPHA:  TERM_TRACK
          ALPHA:  TRACK_INITIATE
          ALPHA:  VALIDATE_HEADER
          DATA:  COMMAND_ID
          DATA:  FOUND
          DATA:  HO_ID
          DATA:  IMAGE_ID
          ENTITY_CLASS:  IMAGE
          EVENT:  ALLOCATE
          EVENT:  SCHEDULE
          EVENT:  SUMMARIZE
          INPUT_INTERFACE:  CC_IN
          OUTPUT_INTERFACE:  CC_OUT
          OUTPUT_INTERFACE:  DATA_RECORD
          SUBNET:  RECORD_DROP
          VALIDATION_POINT:  C2_IMAGE_HANDOVER
          VALIDATION_POINT:  STARTING_POINT.
     ENABLED BY:
          INPUT_INTERFACE:  CC_IN.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
     STRUCTURE:
        INPUT_INTERFACE:  CC_IN
        ALPHA:  VALIDATE_HEADER
        DO
           ALPHA:  ACKNOWLEDGE
```

BEST AVAILABLE COPY

```
            OUTPUT_INTERFACE:  CC_OUT
        AND
            CONSIDER  DATA:  COMMAND_ID
            IF (HANDOVER_IMAGE)
               ALPHA:  TRACK_INITIATE
               VALIDATION_POINT:  C2_IMAGE_HANDOVER
               EVENT:  ALLOCATE
               OUTPUT_INTERFACE:  DATA_RECORD
            OR (INITIATE_ENGAGEMENT_MODE)
               ALPHA:  STARTER
               VALIDATION_POINT:  STARTING_POINT
               ALPHA:  ENGAGEMENT_INITIATION
               EVENT:  SCHEDULE
               EVENT:  SUMMARIZE
               TERMINATE
            OR (TERMINATE_ENGAGEMENT_MODE)
               ALPHA:  TERM_ENGAGEMENT
               TERMINATE
            OR (DROP_TRACK)
               SELECT ENTITY_CLASS:  IMAGE SUCH THAT (IMAGE_ID=HO_ID)
               IF (FOUND)
                  SUBNET:  RECORD_DROP
                  ALPHA:  TERM_TRACK
                  OUTPUT_INTERFACE:  DATA_RECORD
               OTHERWISE
                  ALPHA:  CC_ERROR_PROCESSING
                  TERMINATE
               END
            OR (CC_MESSAGE_ERROR)
               ALPHA:  CC_ERROR_PROCESSING
               TERMINATE
            END
         END
      END.


R_NET:  CONTROL_RESOURCES.
      DESCRIPTION:
            "THE TLS_OPS SHALL IMPLEMENT THE REQUIREMENTS SPECIFIED
             IN THE OPSPR, REFERENCE 2.2, ASSOCIATED WITH MANAGEMENT
             AND CONTROL OF TLS RESOURCES AND SHALL PERFORM THE
             FUNCTIONS HEREIN DEFINED AND DIAGRAMMED IN THE CONTRES
             R_NET.
               THE OPS SHALL ASSIGN TRACK RATES AND ENERGY ALLOWANCES
                  TO EACH IMAGE HANDED OVER FROM COMMAND AND CONTROL
                  AND SHALL DETERMINE ENERGY BOUNDS AND TRACK RATE
                  ENVELOPES FOR EACH HANDOVER IMAGE WHICH REMAINS IN
                  TRACK STATUS. THE ENERGY BOUNDS AND TRACK RATE
                  ENVELOPES SHALL BE MAINTAINED WITHIN AN ACCURACY
                  AND RESOLUTION TOLERANCE SUFFICIENT TO MEET THE
                  SPECIFIED REQUIREMENTS FOR DETERMINATION OF TARGET
                  INTERCEPT CONDITIONS.
               THE OPS SHALL ASSESS STATUS OF THE TLS RESOURCES AND
                  SHALL ALLOCATE TLS RESOURCES TO EACH HANDOVER IMAGE
                  BASED ON RADAR SUBSYSTEM PERFORMANCE CAPABILITIES
                  AND SHALL MAINTAIN A GRACEFUL DEGRADATION POSTURE
                  WHILE UNDER OVERLOAD CONDITIONS.
               THE OPS SHALL GENERATE TLS RESOURCE UTILIZATION
```

```
                    PROFILES AND SHALL COMMIT THESE DATA TO PERMANENT
                    FILE THROUGH THE DATA RECORD OUTPUT INTERFACE.".
        REFERS TO:
              ALPHA:  ASSIGN_RATE
              ALPHA:  CREATE_STATE_FILE
              ALPHA:  DROP_LAST
              ALPHA:  MAKE_ALLOCATION
              ENTITY_TYPE:  IMAGE_IN_TRACK
              ENTITY_TYPE:  LAST_PULSE
              ENTITY_TYPE:  RETURNED_PULSE
              SUBNET:  TALLY_RETURNS.
        ENABLED BY:
              EVENT:  ALLOCATE.
        TRACED FROM:
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.
        STRUCTURE:
           DO
              FOR EACH  ENTITY_TYPE:  RETURNED_PULSE
                 DO SUBNET:  TALLY_RETURNS END
           AND
              FOR EACH  ENTITY_TYPE:  LAST_PULSE
                 DO ALPHA:  DROP_LAST END
           AND
              FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK
                 DO ALPHA:  CREATE_STATE_FILE END
           END
           ALPHA:  MAKE_ALLOCATION
           FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK
              DO ALPHA:  ASSIGN_RATE END
           TERMINATE
        END.


R_NET:  RADAR_SUMMARY.
        REFERS TO:
              ALPHA:  COMPLETE_SUMMARY
              DATA:  MODE
              ENTITY_TYPE:  RETURNED_PULSE
              EVENT:  SUMMARIZE
              OUTPUT_INTERFACE:  DATA_RECORD
              SUBNET:  SUM_RETURNS.
        ENABLED BY:
              EVENT:  SUMMARIZE.
        TRACED FROM:
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
```

```
        TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
    STRUCTURE:
       CONSIDER  DATA:  MODE
       IF (ENGAGED)
          FOR EACH  ENTITY_TYPE:  RETURNED_PULSE
             DO SUBNET:  SUM_RETURNS END
          ALPHA:  COMPLETE_SUMMARY
          EVENT:  SUMMARIZE
          OUTPUT_INTERFACE:  DATA_RECORD
       OR (STANDBY)
          TERMINATE
       END
    END.


R_NET:  RADAR_TIMING.
    DESCRIPTION:
          "RADAR_TIMING MAINTAINS A RECORD OF THE
           RADAR_CLOCK_TIME.".
    REFERS TO:
         ALPHA:  UPDATE_RADAR_CLOCK
         INPUT_INTERFACE:  RADAR_CLOCK_IN.
    ENABLED BY:
         INPUT_INTERFACE:  RADAR_CLOCK_IN.
    STRUCTURE:
       INPUT_INTERFACE:  RADAR_CLOCK_IN
       ALPHA:  UPDATE_RADAR_CLOCK
       TERMINATE
    END.


R_NET:  RESPONSE_TO_RADAR.
    DESCRIPTION:
          "THE TLS_DPS SHALL IMPLEMENT THE REQUIREMENTS SPECIFIED
           IN THE TLS_RADAR_DPS_INTERFACE_SPECIFICATION ASSOCIATED
           WITH PROCESSING RADAR SUBSYSTEM RESPONSES TO COMMANDS,
           AND SHALL PERFORM THE FUNCTIONS HEREIN DEFINED AND
           DIAGRAMMED IN THE RESPRAD R_NET.
             THE DPS SHALL RECEIVE AND PROCESS RADAR MESSAGES
               TRANSMITTED BY THE TLS RADAR SUBSYSTEM AND SHALL
               INTERROGATE EACH MESSAGE FOR ERRORS AND SHALL
               PROCESS ALL DETECTED MESSAGE ERRORS.
             THE DPS SHALL, UPON RECEIPT OF ERROR_FREE RADAR
               MESSAGES, DETECT AND PROCESS REDUNDANT_IMAGES,
               GHOST_IMAGES, AND LOW_ELEVATION_IMAGES, AND SHALL
               UPDATE STATE_PARAMETERS FOR EACH IMAGE_IN_TRACK.
             THE DPS SHALL TERMINATE TRACK ON EACH IMAGE WHICH IS
               DETERMINED TO BE EITHER A REDUNDANT OR GHOST IMAGE
               OR WHICH IS FOUND TO EXCEED THE LOW_ELEVATION
               CONSTRAINTS AND SHALL MAINTAIN TRACK ON EACH IMAGE
               DETERMINED TO BE REAL.
             THE DPS SHALL CONSTRUCT AND MAINTAIN DESCRIPTIVE DATA
               FILES FOR EACH IMAGE WHICH IS EITHER MAINTAINED IN
               TRACK OR DROPPED FROM TRACK AND SHALL PERIODICALLY
               COMMIT THESE DATA TO PERMANENT RECORDS THROUGH THE
               DATA RECORDS INTERFACE.".
    IMPLEMENTS:
         VERSION:  PDL2_COMPILER_LIMITATION_COMPENSATION.
```


BEST AVAILABLE COPY

```
REFERS TO:
      ALPHA:  FORM_UPDATE
      ALPHA:  GHOST_DETERMINATION
      ALPHA:  GHOST_TERMINATION
      ALPHA:  LOW_ELEVATION_DETERMINATION
      ALPHA:  LOW_TERMINATION
      ALPHA:  REDUN_DETERMINATION
      ALPHA:  REDUN_TERMINATION
      ALPHA:  REMEMBER_STOP
      ALPHA:  RR_ERROR_PROCESSING
      ALPHA:  UPDATE_STATE
      DATA:  FOUND
      DATA:  GHOST_IMAGE
      DATA:  IMAGE_ID
      DATA:  LOW_ELEVATION
      DATA:  PULSE_ID
      DATA:  PULSE_TYPE
      DATA:  RADAR_TYPE
      DATA:  REDUNDANT_IMAGE
      DATA:  RR_ORDER_ID
      DATA:  TARGET_ID
      DATA:  VALID_RETURN
      ENTITY_CLASS:  IMAGE
      ENTITY_CLASS:  PULSE
      ENTITY_TYPE:  IMAGE_IN_TRACK
      INPUT_INTERFACE:  RADAR_IN
      OUTPUT_INTERFACE:  DATA_RECORD
      SUBNET:  EXTRACT_MEASUREMENT
      SUBNET:  MISSING_RETURNS
      SUBNET:  RECORD_DRAW.
ENABLED BY:
      INPUT_INTERFACE:  RADAR_IN.
TRACED FROM:
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
```


BEST AVAILABLE COPY

```
      TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
STRUCTURE:
   INPUT_INTERFACE:  RADAR_IN
   SELECT ENTITY_CLASS:  PULSE SUCH THAT (PULSE_ID=RR_ORDER_ID)
   IF (FOUND)
      IF (RADAR_TYPE=PULSE_TYPE)
         DO
            ALPHA:  REMEMBER_STOP
            FOR EACH  ENTITY_CLASS:  PULSE
               DO SUBNET:  MISSING_RETURNS END
            SELECT ENTITY_CLASS:  PULSE SUCH THAT
            (PULSE_ID=RR_ORDER_ID)
            TERMINATE
         AND
            SUBNET:  EXTRACT_MEASUREMENT
            IF (VALID_RETURN)
               ALPHA:  UPDATE_STATE
               DO
                  ALPHA:  LOW_ELEVATION_DETERMINATION
                  IF (LOW_ELEVATION)
                     SUBNET:  RECORD_DROP
                     ALPHA:  LOW_TERMINATION
                     OUTPUT_INTERFACE:  DATA_RECORD
                  OTHERWISE
                     TERMINATE
                  END
               AND
                  ALPHA:  FORM_UPDATE
                  OUTPUT_INTERFACE:  DATA_RECORD
               AND
                  FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK
                  SUCH THAT (IMAGE_ID<>TARGET_ID)
                     DO ALPHA:  REDUN_DETERMINATION END
                  SELECT ENTITY_CLASS:  IMAGE SUCH THAT
                  (IMAGE_ID=TARGET_ID)
                  IF (REDUNDANT_IMAGE)
                     SUBNET:  RECORD_DROP
                     ALPHA:  REDUN_TERMINATION
                     OUTPUT_INTERFACE:  DATA_RECORD
                  OTHERWISE
                     TERMINATE
                  END
               END
            OTHERWISE
               ALPHA:  GHOST_DETERMINATION
               IF (GHOST_IMAGE)
                  SUBNET:  RECORD_DROP
                  ALPHA:  GHOST_TERMINATION
                  OUTPUT_INTERFACE:  DATA_RECORD
               OTHERWISE
                  TERMINATE
               END
            END
         END
      OTHERWISE
         ALPHA:  RR_ERROR_PROCESSING
         TERMINATE
      END
```


BEST AVAILABLE COP

```
      OTHERWISE
         ALPHA:  RR_ERROR_PROCESSING
         TERMINATE
      END
   END.


R_NET:  SKED_R.
   DESCRIPTION:
         "SKED_R CONSTRUCTS THE ORDERED FILE OF DATA FOR
          A FRAME.".
   REFERS TO:
         ALPHA:  INITIALIZE_SKED_R
         ALPHA:  PICK_CANDIDATES
         DATA:  LAST_PULSE
         DATA:  MODE
         DATA:  TEOF
         DATA:  TRACK_RATE
         ENTITY_TYPE:  IMAGE_IN_TRACK
         EVENT:  XRB
         FILE:  CANDIDATE
         SUBNET:  FORM_FRAME.
   ENABLED BY:
         EVENT:  SCHEDULE.
   TRACED FROM:
         DECISION:  RADAR_SCHEDULER_PRIORITIZATION
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
   STRUCTURE:
      CONSIDER  DATA:  MODE
      IF (ENGAGED)
         ALPHA:  INITIALIZE_SKED_R
         FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK SUCH THAT
         (LAST_PULSE+(1.0/TRACK_RATE)<TEOF)
            DO ALPHA:  PICK_CANDIDATES END
         FOR EACH  FILE:  CANDIDATE RECORD
            DO SUBNET:  FORM_FRAME END
         EVENT:  XRB
         TERMINATE
      OR (STANDBY)
         TERMINATE
      END
```

```
END.


R_NET:  XMIT_R.
     DESCRIPTION:
          "XMIT_R BUILDS AND FORWARDS TO THE OUTPUT_INTERFACE
           RADAR_OUT THE COMMANDS OF THE FRAME.".
     REFERS TO:
          ALPHA:  FORM_T1_T2
          ALPHA:  FORM_T3
          ALPHA:  PICK_COMMAND
          ALPHA:  SELECT_COMMAND
          DATA:  RADAR_TYPE
          DATA:  RECORD_FOUND
          EVENT:  SCHEDULE
          EVENT:  XRB
          OUTPUT_INTERFACE:  RADAR_OUT
          VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
     ENABLED BY:
          EVENT:  XRB.
     TRACED FROM:
          DECISION:  RADAR_SCHEDULER_PRIORITIZATION
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
     STRUCTURE:
        ALPHA:  SELECT_COMMAND
        IF (RECORD_FOUND)
           ALPHA:  PICK_COMMAND
           EVENT:  XRB
           CONSIDER  DATA:  RADAR_TYPE
           IF (T3)
              ALPHA:  FORM_T3
           OR (T1 OR T2)
              ALPHA:  FORM_T1_T2
           END
           VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT
           OUTPUT_INTERFACE:  RADAR_OUT
        OTHERWISE
           EVENT:  SCHEDULE
           TERMINATE
        END
     END.
```

```
SUBNET:  EXTRACT_MEASUREMENT.
     IMPLEMENTS:
          VERSION:  PDL2_COMPILER_LIMITATION_COMPENSATION.
     REFERS TO:
          ALPHA:  SET_PULSE
          ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
          ALPHA:  T3_MEASUREMENT_EXTRACTION
          DATA:  IMAGE_ID
          DATA:  TARGET_ID
          ENTITY_CLASS:  IMAGE
          ENTITY_CLASS:  PULSE
          ENTITY_TYPE:  LAST_PULSE
          ENTITY_TYPE:  RETURNED_PULSE
          ENTITY_TYPE:  T1_T2_PULSE
          ENTITY_TYPE:  T3_PULSE.
     REFERRED BY:
          R_NET:  RESPONSE_TO_RADAR.
     STRUCTURE:
        SELECT ENTITY_CLASS:  IMAGE SUCH THAT (IMAGE_ID=TARGET_ID)
        CONSIDER  ENTITY_CLASS:  PULSE
        IF (T3_PULSE)
           ALPHA:  T3_MEASUREMENT_EXTRACTION
        OR (T1_T2_PULSE)
           ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
        OR (LAST_PULSE OR RETURNED_PULSE)
        END
        ALPHA:  SET_PULSE
        RETURN
     END.


SUBNET:  FORM_FRAME.
     DESCRIPTION:  "FORM_FRAME PROVIDES RADAR CONFLICT RESOLUTION.".
     REFERS TO:
          ALPHA:  FIND_CONFLICT
          ALPHA:  MAKE_COMMAND
          ALPHA:  SET_LAST
          DATA:  CANDIDATE_IMAGE_ID
          DATA:  DROP_FLAG
          DATA:  IMAGE_ID
          ENTITY_TYPE:  IMAGE_IN_TRACK.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  SKED_R.
     STRUCTURE:
        ALPHA:  FIND_CONFLICT
```

```
      IF (NOT DROP_FLAG)
         ALPHA:  MAKE_COMMAND
         SELECT ENTITY_TYPE:  IMAGE_IN_TRACK SUCH THAT
         (IMAGE_ID=CANDIDATE_IMAGE_ID)
         (*MUST SUCCEED SINCE SELECTED ON CALLING NET*)
         ALPHA:  SET_LAST
      OTHERWISE
      END
      RETURN
   END.


SUBNET:  MISSING_RETURNS.
   REFERS TO:
         ALPHA:  SET_LAST
         DATA:  RECEIVE_STOP
         DATA:  STOP_TIME
         ENTITY_CLASS:  PULSE
         ENTITY_TYPE:  LAST_PULSE
         ENTITY_TYPE:  RETURNED_PULSE
         ENTITY_TYPE:  T1_T2_PULSE
         ENTITY_TYPE:  T3_PULSE.
   TRACED FROM:
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS.
   REFERRED BY:
         R_NET:  RESPONSE_TO_RADAR.
   STRUCTURE:
      CONSIDER  ENTITY_CLASS:  PULSE
      IF (T1_T2_PULSE)
         IF (RECEIVE_STOP<STOP_TIME)
            ALPHA:  SET_LAST
         OTHERWISE
         END
      OR (T3_PULSE)
         IF (RECEIVE_STOP<STOP_TIME)
            ALPHA:  SET_LAST
         OTHERWISE
         END
      OR (LAST_PULSE OR RETURNED_PULSE)
      END
      RETURN
   END.


SUBNET:  RECORD_DROP.
   REFERS TO:
         ALPHA:  SET_DROP
         ENTITY_CLASS:  IMAGE
         ENTITY_TYPE:  DROPPED_IMAGE
         ENTITY_TYPE:  IMAGE_IN_TRACK
         EVENT:  ALLOCATE.
   TRACED FROM:
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
   REFERRED BY:
         R_NET:  CC_RESPONSE
         R_NET:  RESPONSE_TO_RADAR.
```

```
      STRUCTURE:
         CONSIDER  ENTITY_CLASS:  IMAGE
         IF (IMAGE_IN_TRACK)
            ALPHA:  SET_DROP
            EVENT:  ALLOCATE
         OR (DROPPED_IMAGE)
         END
         RETURN
      END,


SUBNET:  SUM_RETURNS,
      REFERS TO:
            ALPHA:  DROP_PULSE
            ALPHA:  SET_SUMMED
            ALPHA:  SUM_USAGE
            DATA:  ACCOUNTED_FOR.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  RADAR_SUMMARY.
      STRUCTURE:
         CONSIDER  DATA:  ACCOUNTED_FOR
         IF (COUNTED)
            ALPHA:  SUM_USAGE
            ALPHA:  DROP_PULSE
         OR (NEITHER)
            ALPHA:  SUM_USAGE
            ALPHA:  SET_SUMMED
         OR (SUMMED)
         END
         RETURN
      END,


SUBNET:  TALLY_RETURNS,
      REFERS TO:
            ALPHA:  DROP_PULSE
            ALPHA:  SET_COUNTED
            ALPHA:  SUM_ENERGY
            DATA:  ACCOUNTED_FOR.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  CONTROL_RESOURCES.
      STRUCTURE:
         CONSIDER  DATA:  ACCOUNTED_FOR
         IF (SUMMED)
            ALPHA:  SUM_ENERGY
            ALPHA:  DROP_PULSE

         OR (NEITHER)
            ALPHA:  SUM_ENERGY
            ALPHA:  SET_COUNTED
         OR (COUNTED)
         END
         RETURN
      END,
```

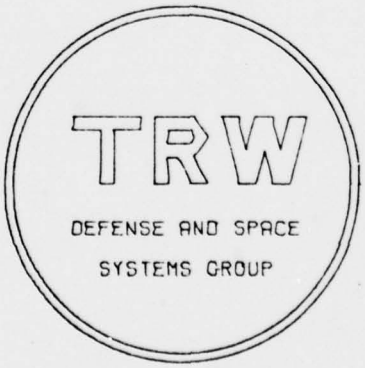

BEST AVAILABLE COPY

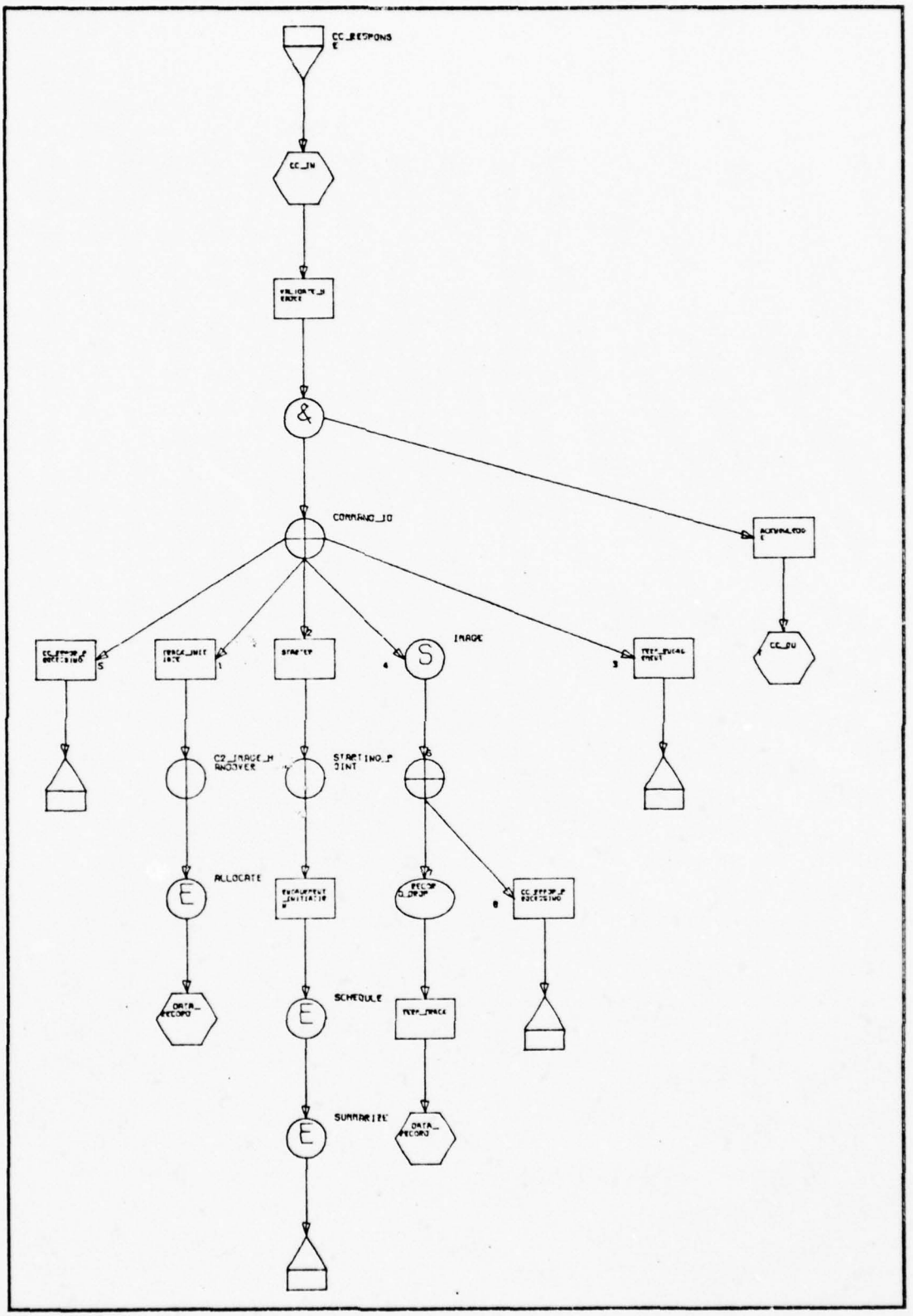
APPENDIX G

TLS REQUIREMENTS NETWORKS

REQUIREMENTS ENGINEERING AND VALIDATION

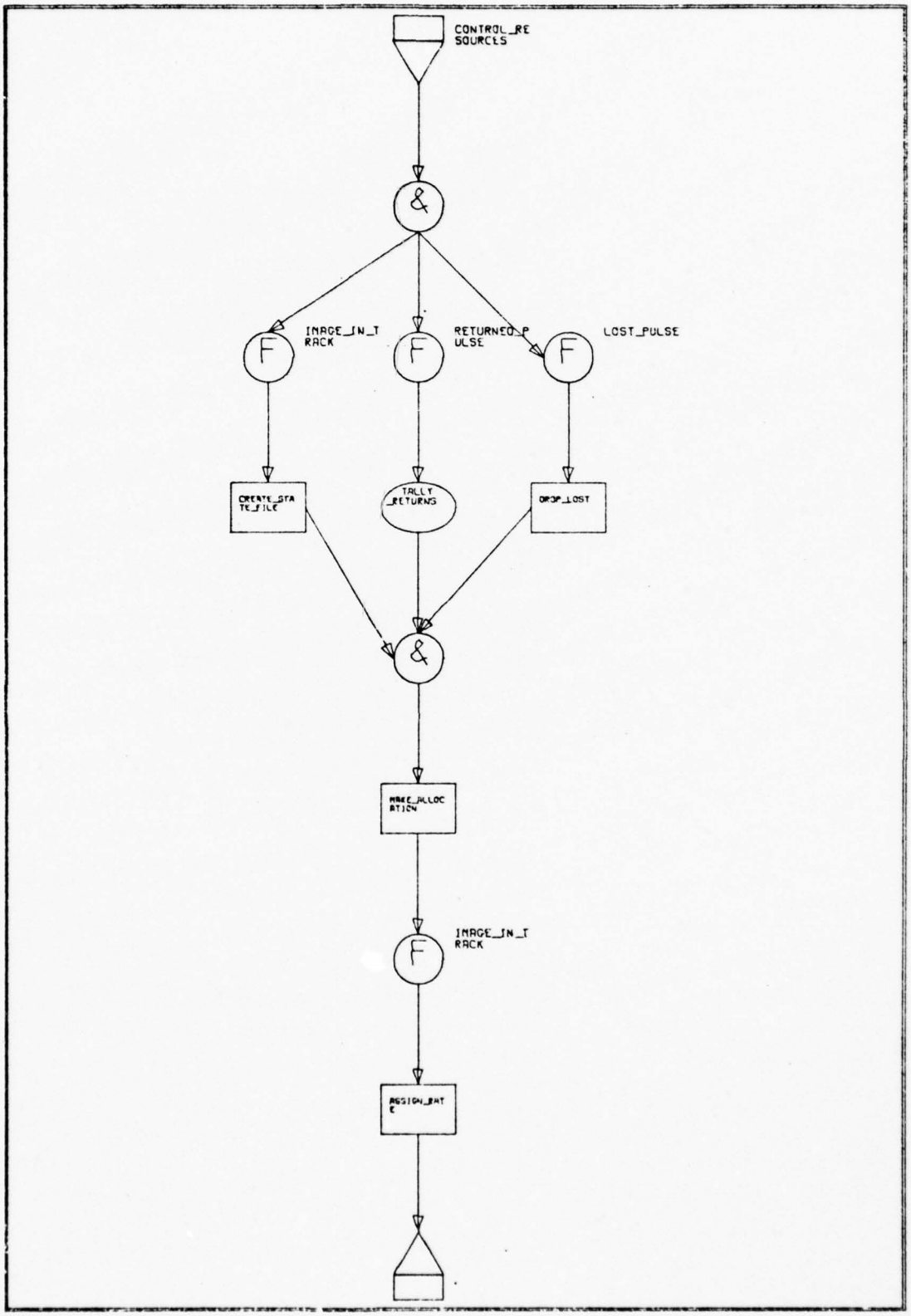
SYSTEM

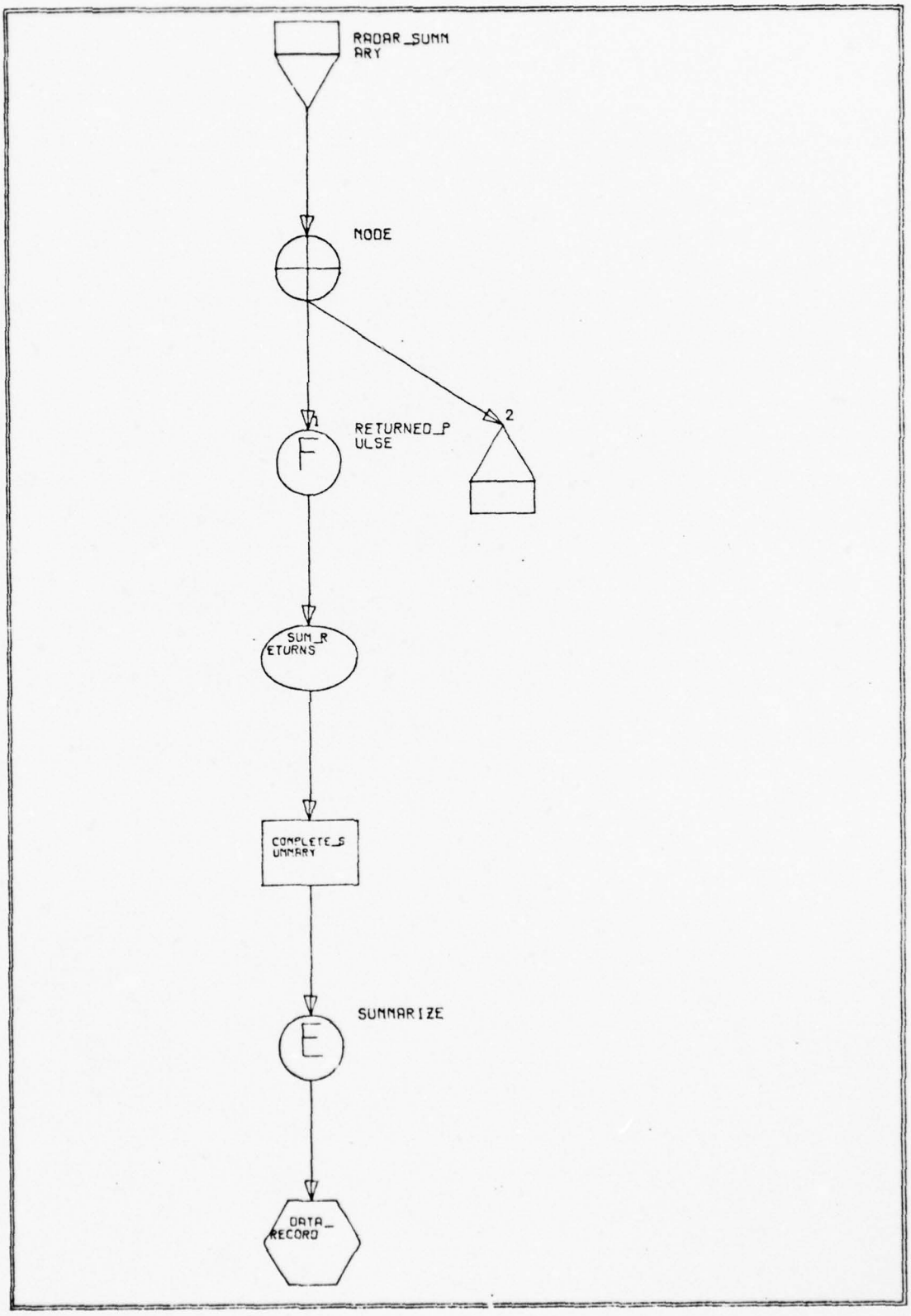



CC_RESPONS
E
CC_IN
&
COMMAND_IO
IMAGE
S
ALLOCATE
SCHEDULE
SUMMARIZE
CC_DU

CC_RESPONSE
STRUCTURE LEGEND

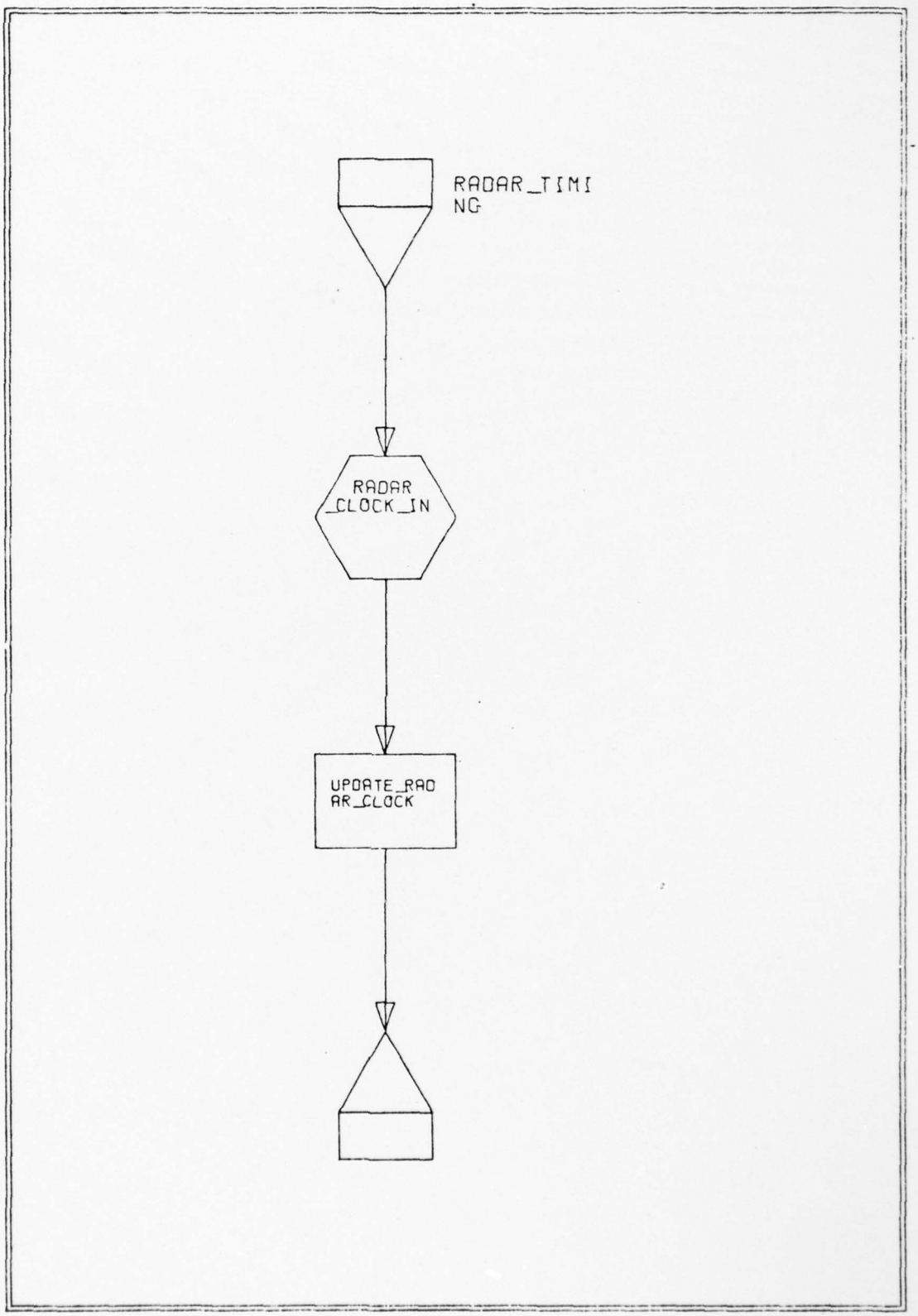
NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(HANDOVER_IMAGE)
2		(INITIATE_ENGAGEMENT_MODE)
3		(TERMINATE_ENGAGEMENT_MODE)
4		(DROP_TRACK)
5		(CC_MESSAGE_ERROR)
6		(IMAGE_ID=HO_ID)
7		(FOUND)
8		OTHERWISE

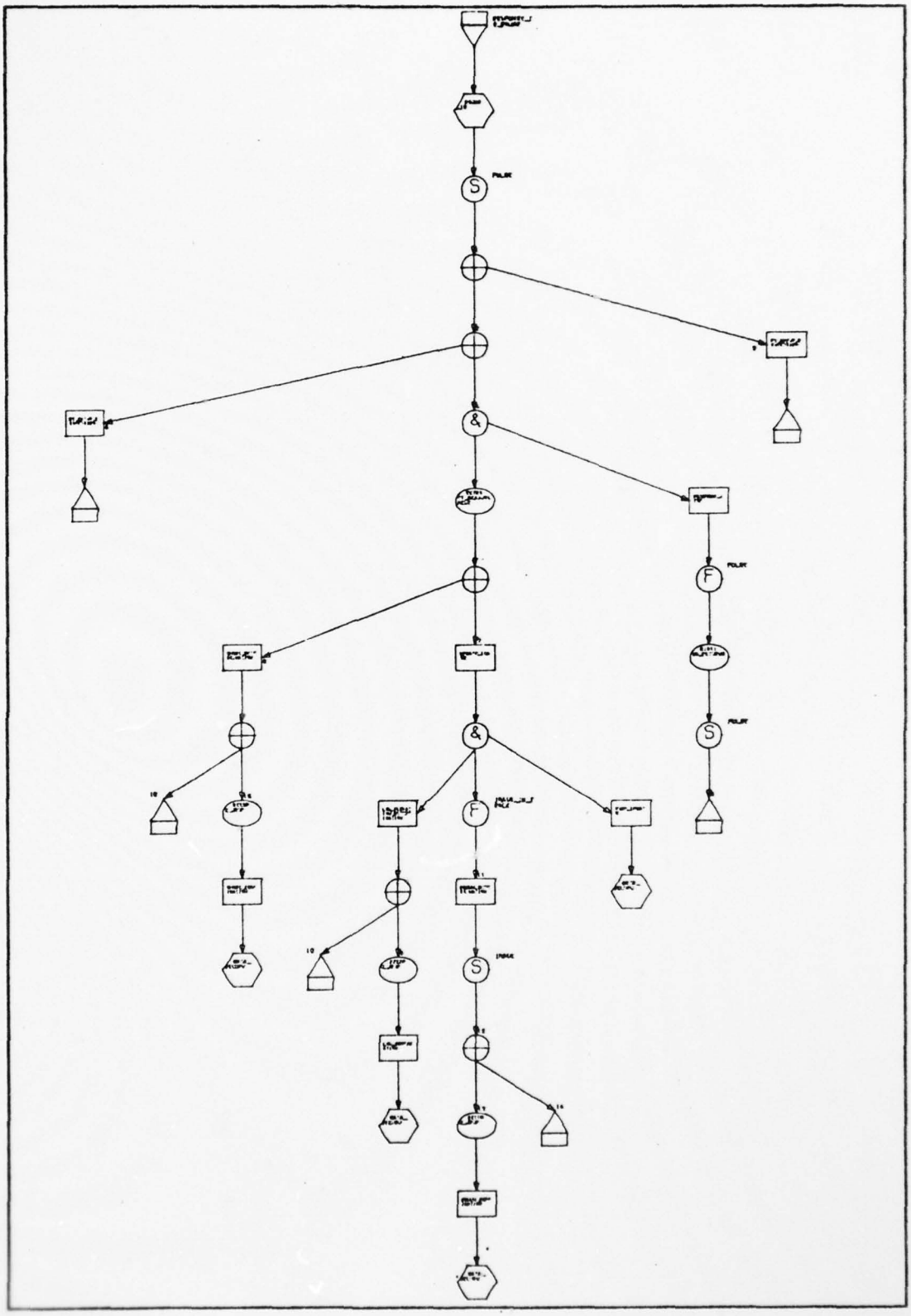

CONTROL_RE
SOURCES
&
F
IMAGE_IN_T
RACK
F
RETURNED_P
ULSE
F
LOST_PULSE
CREATE_STA
TE_FILE
TALLY
_RETURNS
DROP_LOST
&
MAKE_ALLOC
ATION
F
IMAGE_IN_T
RACK
ASSIGN_BAT
E


RADAR_SUMM
ARY
MODE
1
RETURNED_P
ULSE
2
F
SUM_R
ETURNS
COMPLETE_S
UMMARY
SUMMARIZE
E
DATA_
RECORD

RADAR_SUMMARY
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(ENGAGED)
2		(STANDBY)

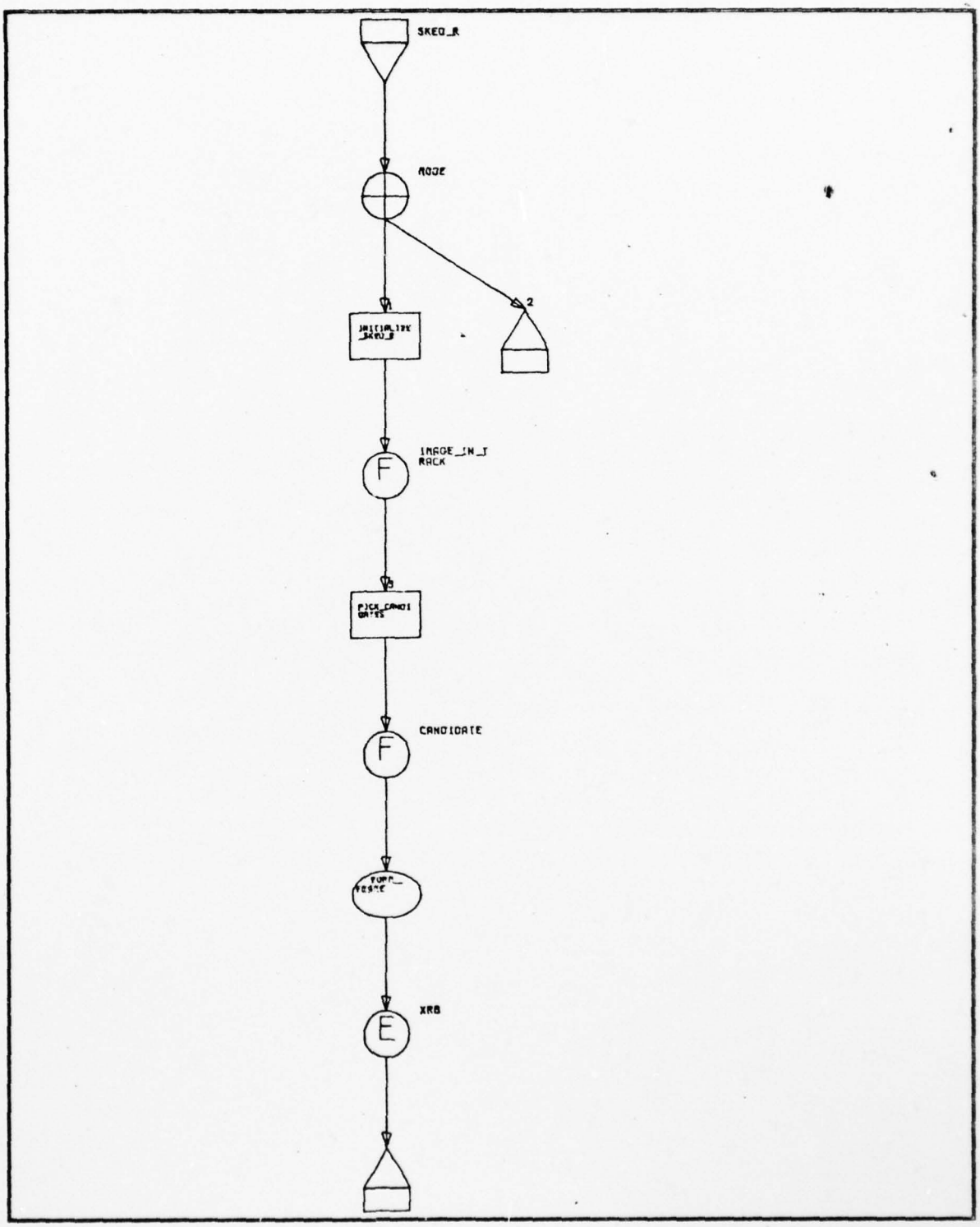

RADAR_TIMI
NG
RADAR
_CLOCK_IN
UPDATE_RAD
AR_CLOCK



BEST AVAILABLE COPY

RESPONSE_TO_RADAR
STRUCTURE LEGEND

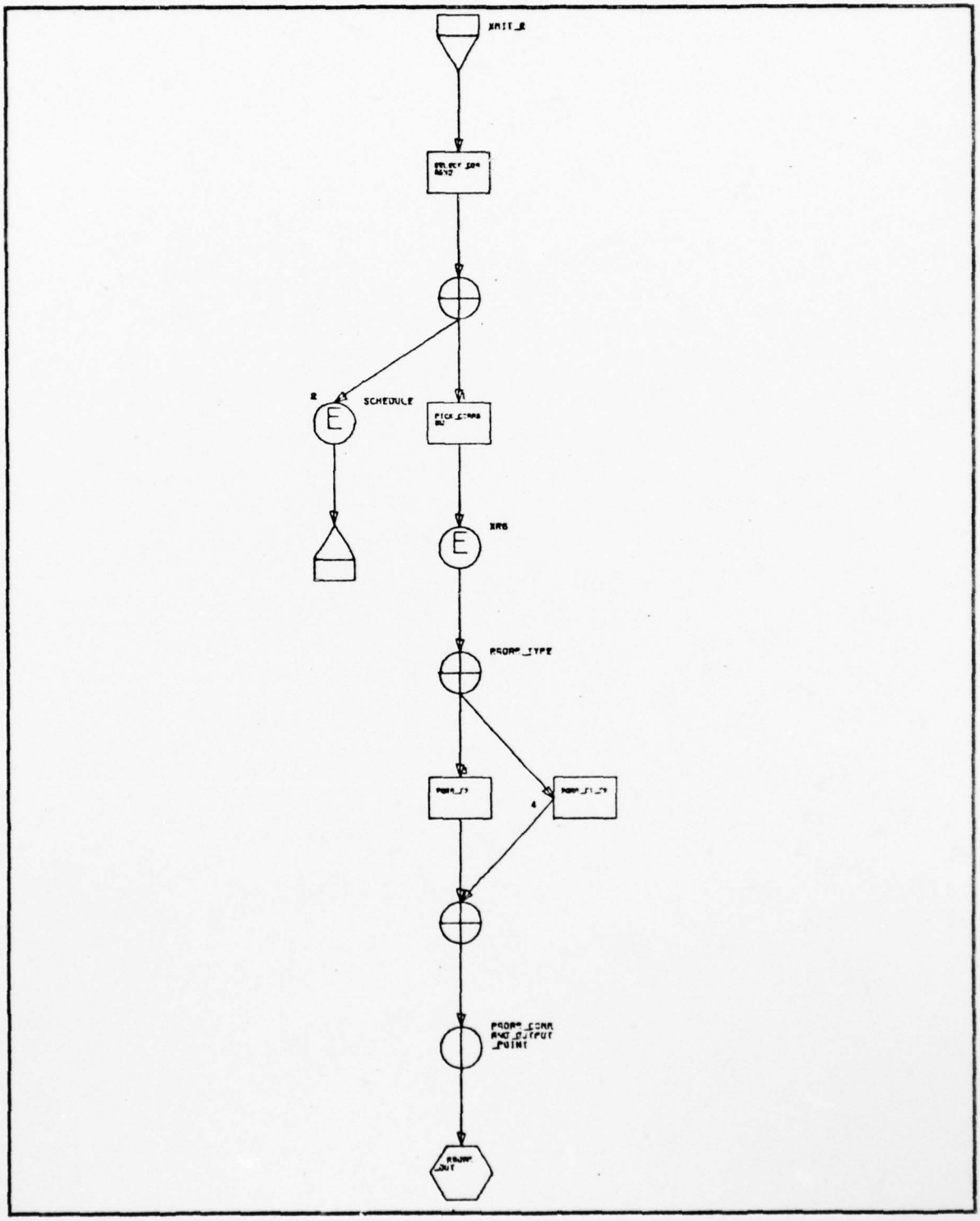
NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(PULSE_ID=RR_ORDER_ID)
2		(FOUND)
3		OTHERWISE
4		(RADAR_TYPE=PULSE_TYPE)
5		OTHERWISE
6		(PULSE_ID=RR_ORDER_ID)
7		(VALID_RETURN)
8		OTHERWISE
9		(LOW_ELEVATION)
10		OTHERWISE
11		(IMAGE_ID<>TARGET_ID)
12		(IMAGE_ID=TARGET_ID)
13		(REDUNDANT_IMAGE)
14		OTHERWISE
15		(GHOST_IMAGE)
16		OTHERWISE



BEST AVAILABLE COPY

SKED_R
STRUCTURE LEGEND

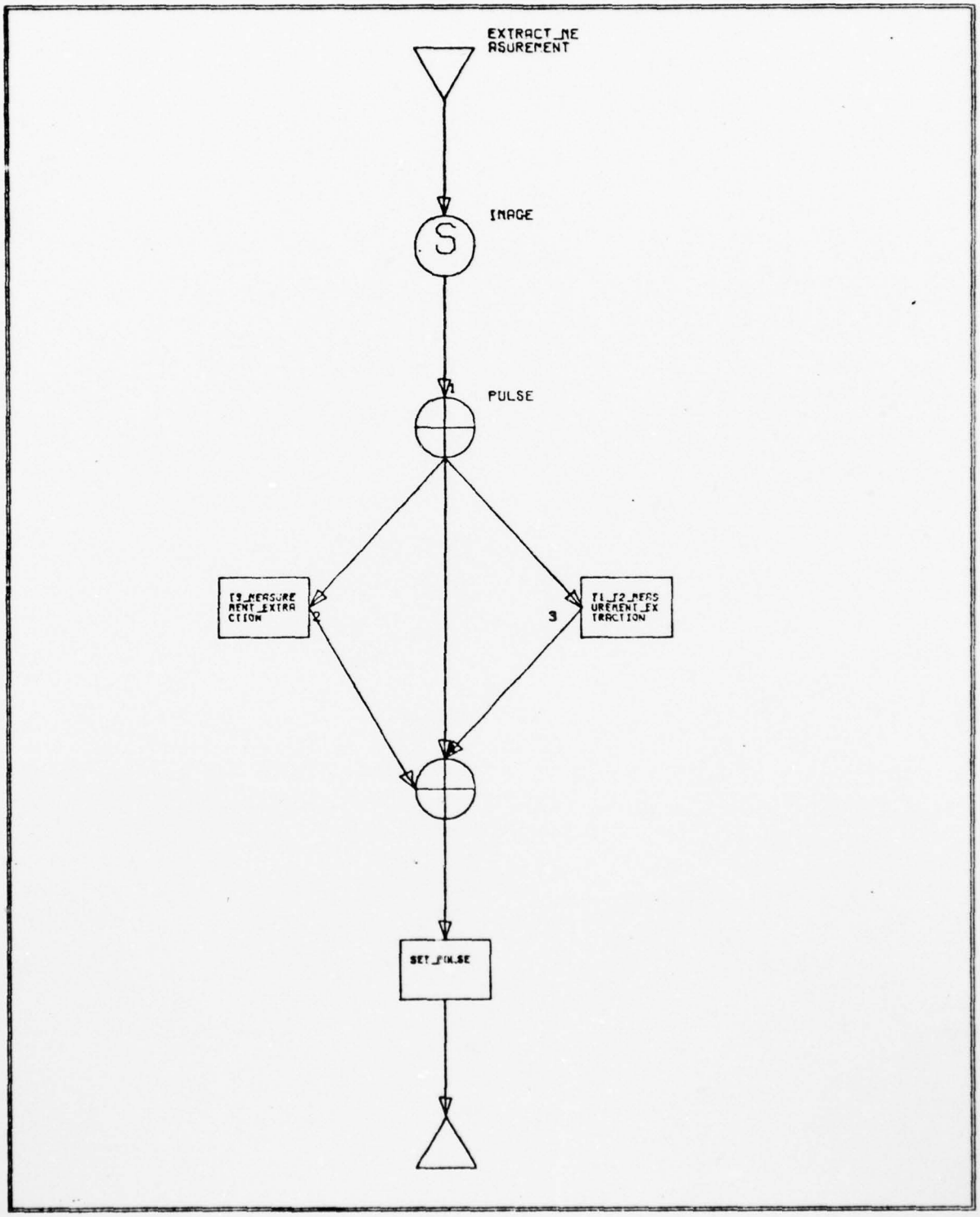
NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(ENGAGED)
2		(STANDBY)
3		(LAST_PULSE+(1.0/TRACK_RATE)<TEOF)



BEST AVAILABLE COPY

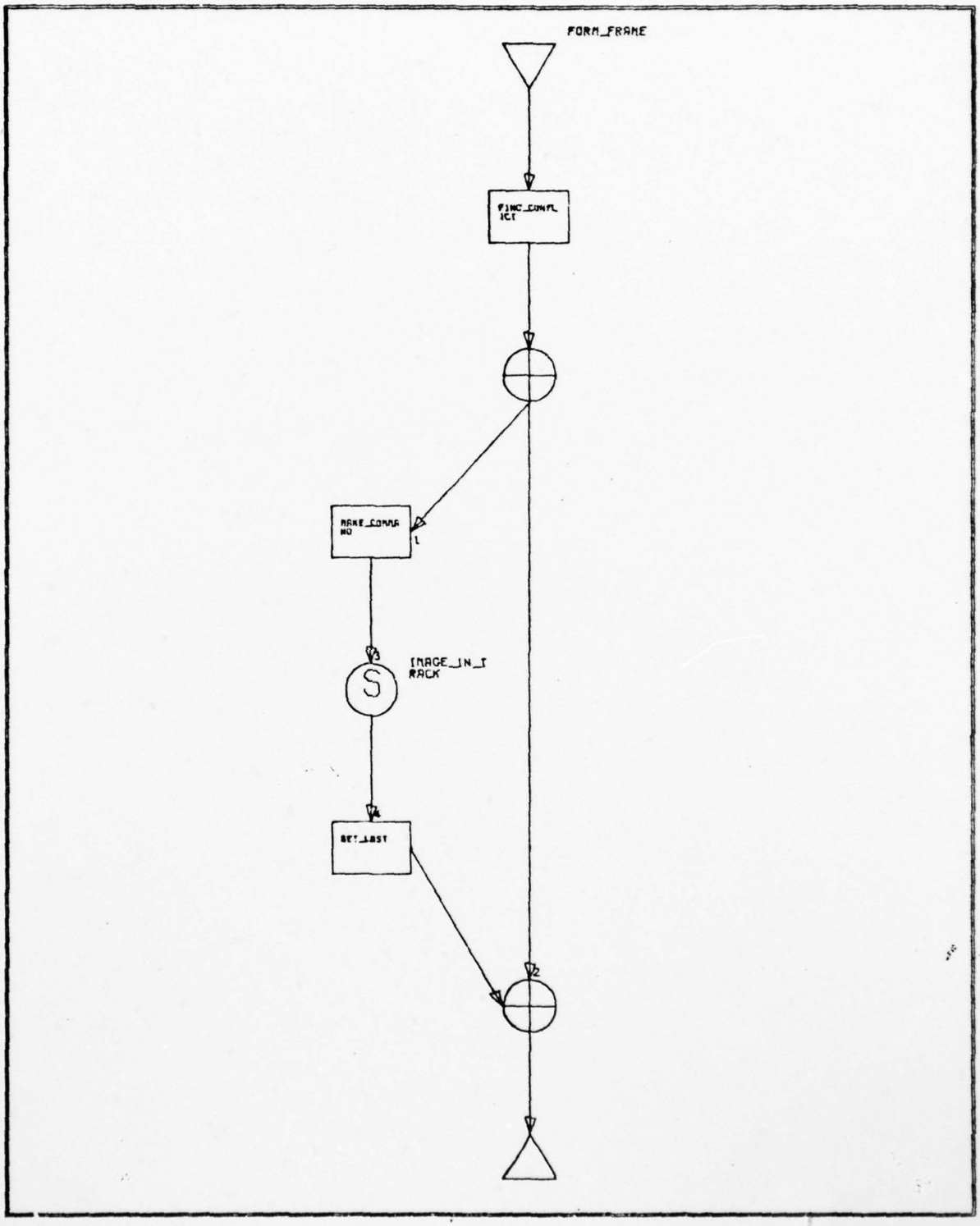
XMIT_R
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(RECORD_FOUND)
2		OTHERWISE
3		(T3)
4		(T1 OR T2)


EXTRACT_ME
ASUREMENT
IMAGE
S
PULSE
T9_MEASURE
MENT_EXTRA
CTION
2
3
T1_T2_MEAS
UREMENT_EX
TRACTION
SET_PULSE

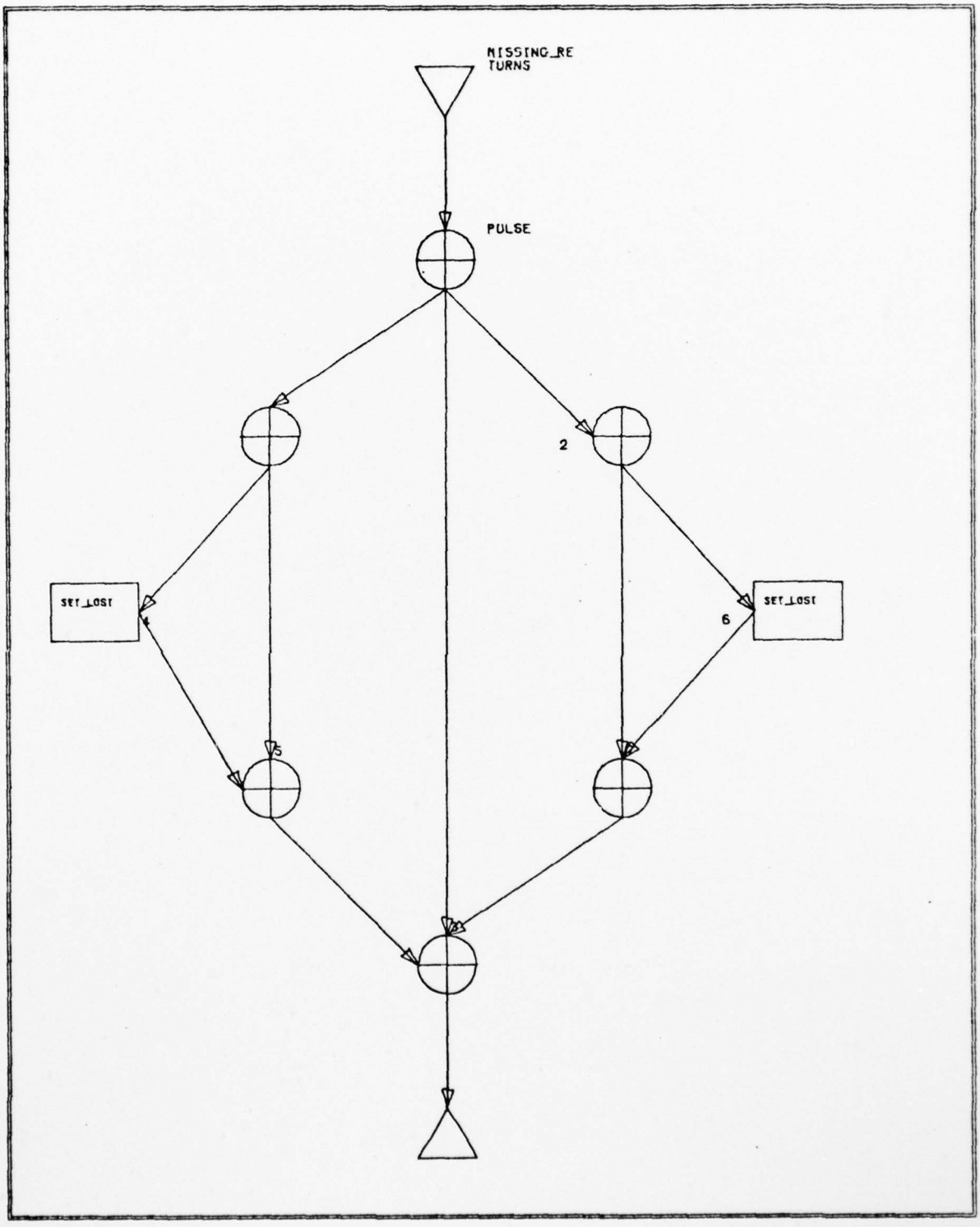
EXTRACT_MEASUREMENT
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(IMAGE_ID=TARGET_ID)
2		(T3_PULSE)
3		(T1_T2_PULSE)
4		(LOST_PULSE OR RETURNED_PULSE)


FORM_FRAME
FUNC_CONFL
ICT
MAKE_COMMA
ND
1
IMAGE_IN_I
RACK
S
GET_LAST
2

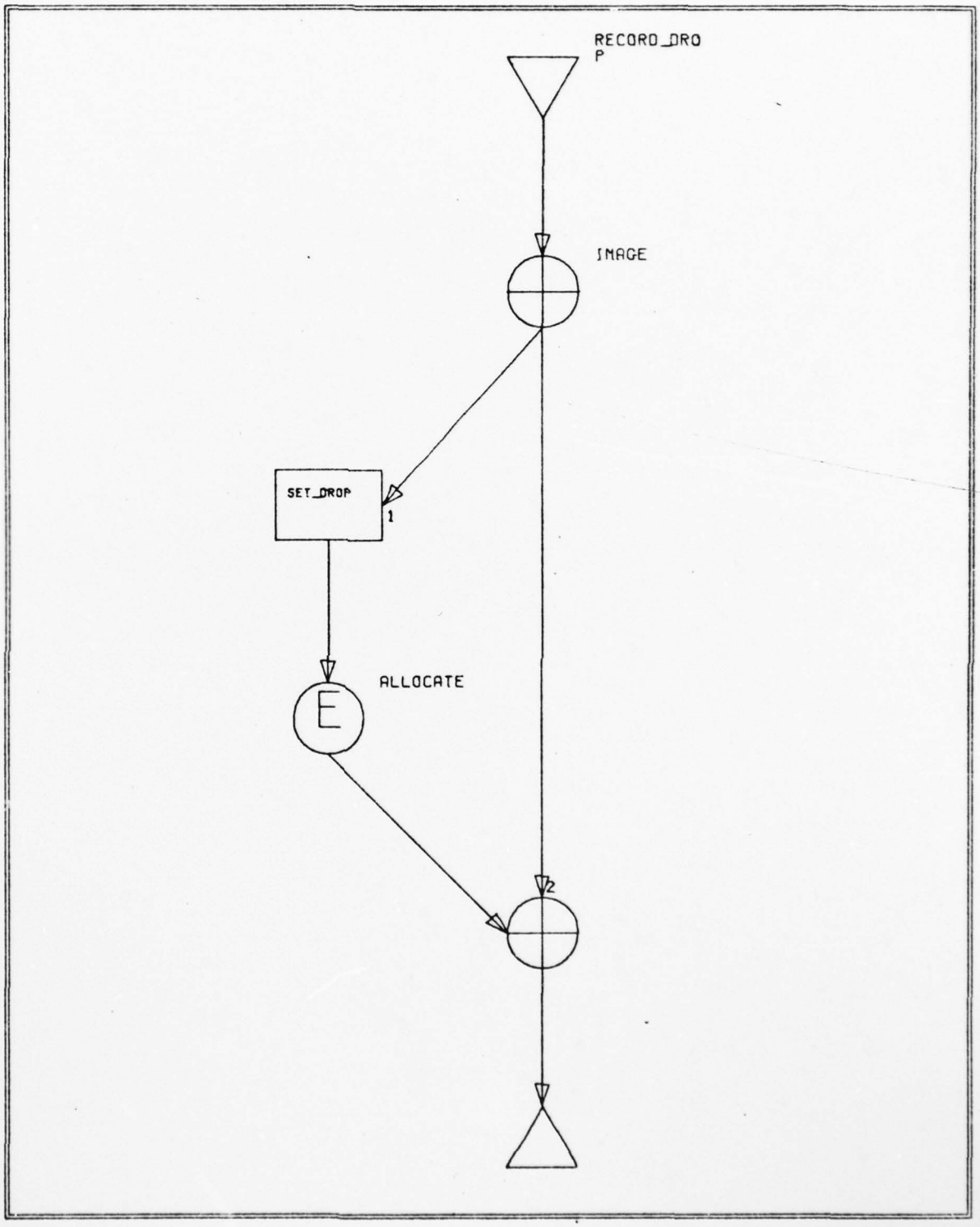
FORM_FRAME
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(NOT DROP_FLAG)
2		OTHERWISE
3		(*MUST SUCCEED SINCE SELECTED ON CALLING NET*)
4		(IMAGE_ID=CANDIDATE_IMAGE_ID)


MISSING_RE
TURNS
PULSE
2
SET_LOST
4
6
SET_LOST
5

MISSING_RETURNS
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(T1_T2_PULSE)
2		(T3_PULSE)
3		(LOST_PULSE OR RETURNED_PULSE)
4		(RECEIVE_STOP<STOP_TIME)
5		OTHERWISE
6		(RECEIVE_STOP<STOP_TIME)
7		OTHERWISE

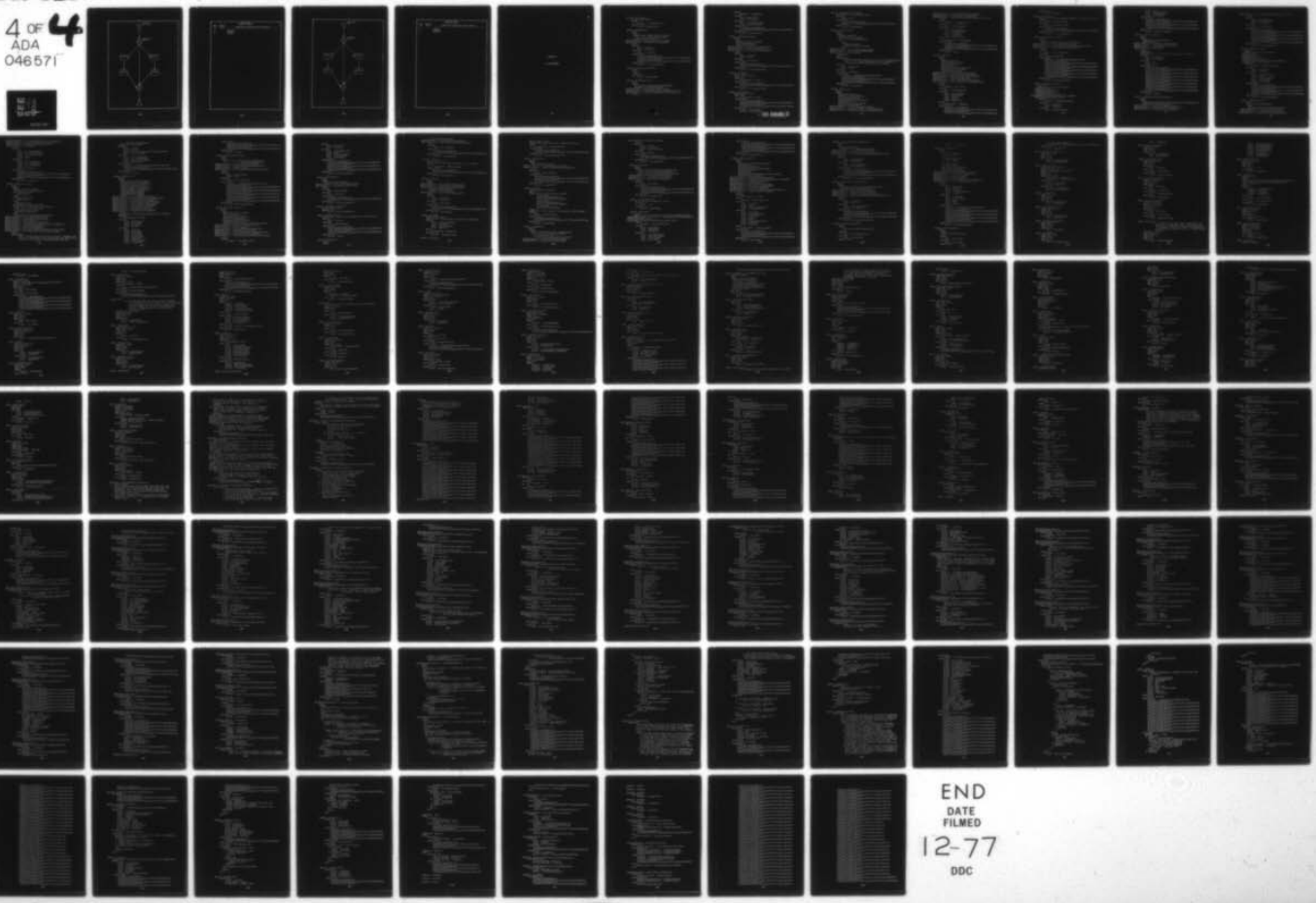

RECORD_DRO
P
IMAGE
SET_DROP
1
ALLOCATE
E
2

RECORD_DROP
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(IMAGE_IN_TRACK)
2		(DROPPED_IMAGE)

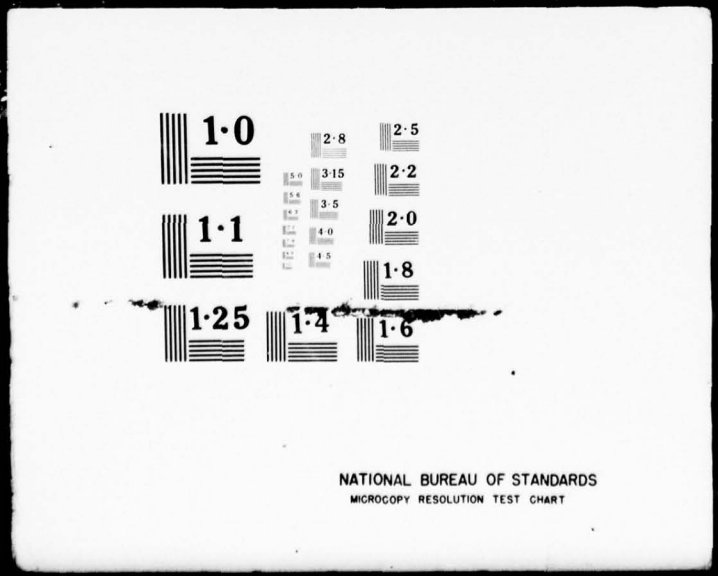
AD-A046 571 TRW DEFENSE AND SPACE SYSTEMS GROUP HUNTSVILLE ALA F/G 9/2
SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY. SREP FINAL REPOR--ETC(U)
AUG 77 M W ALFORD, P H BROWNE, I F BURNS DAS660-75-C-0022
UNCLASSIFIED TRW-27332-6921-026-VOL-1 NL

4 OF 4
ADA
046571

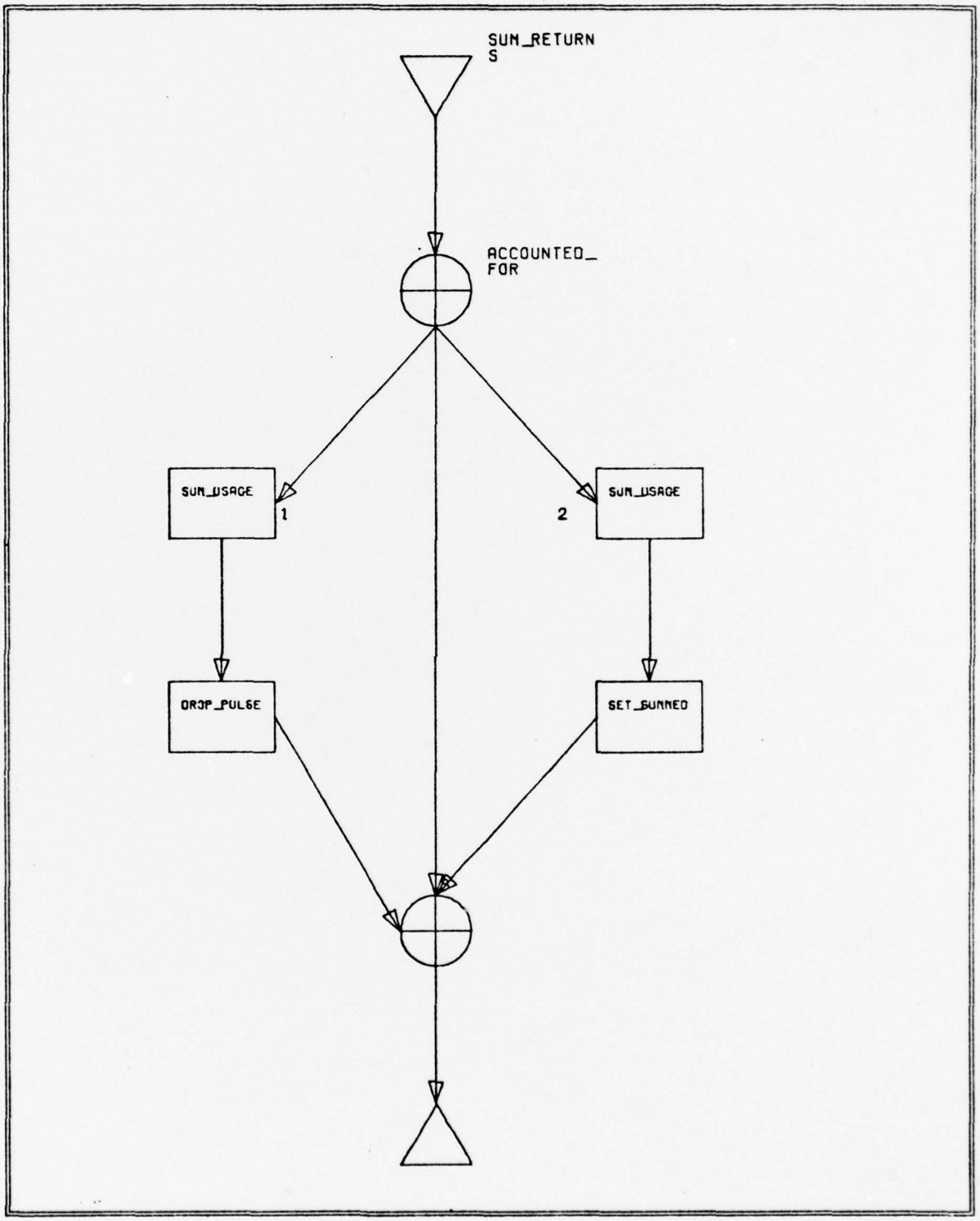


END
DATE FILMED
12-77
DDC

4 OF 4
ADA
046571

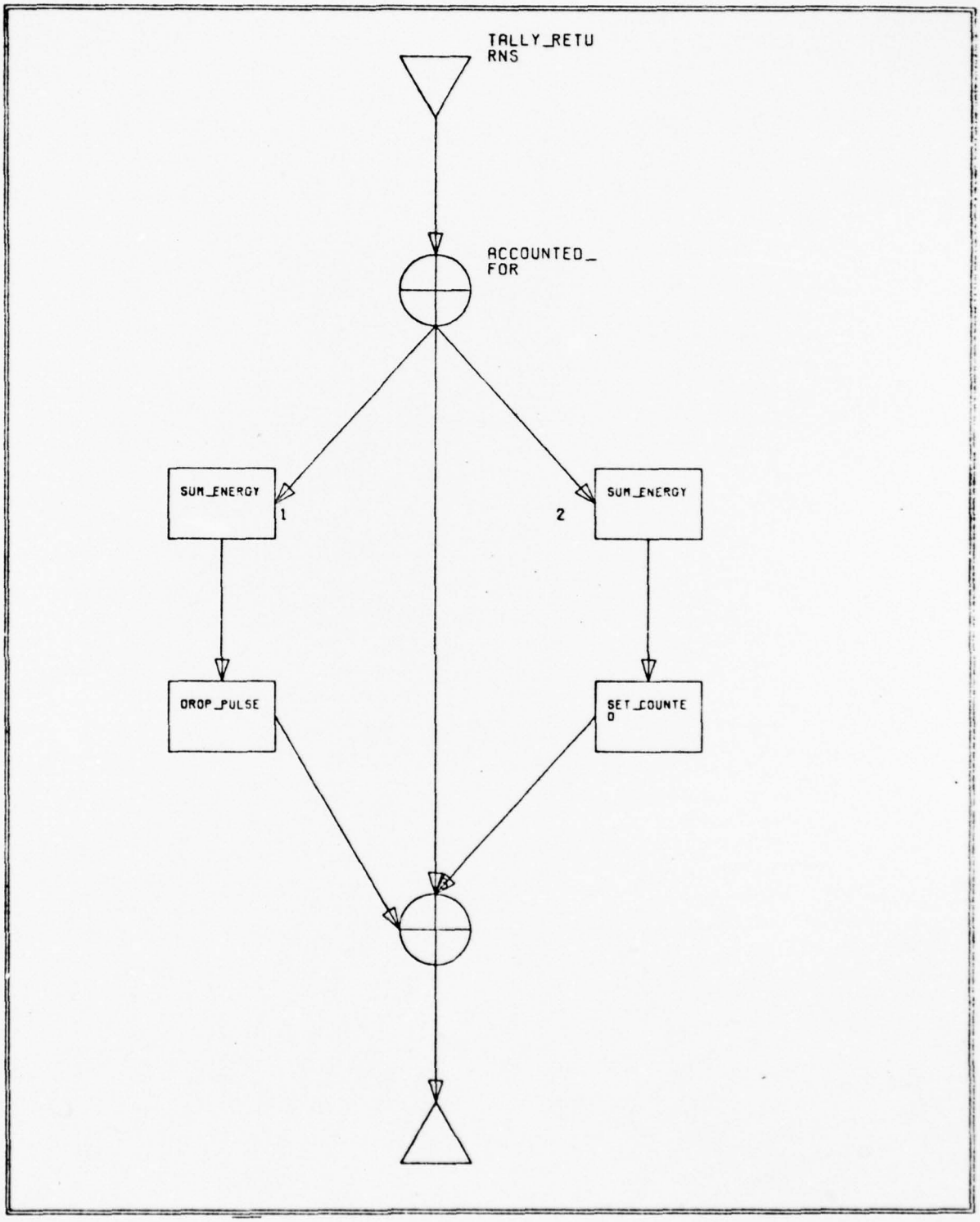



```
WRITELN(OUTPUT,' COMMAND_WAVEFORM=',COMMAND_WAVEFORM);
WRITELN(OUTPUT,' COMMAND_ENERGY=',COMMAND_ENERGY);
WRITELN(OUTPUT,' START_TIME=',START_TIME);
WRITELN(OUTPUT,' IST=',IST);
```


SUM_RETURN
S
ACCOUNTED_
FOR
SUM_USAGE
1
SUM_USAGE
2
DROP_PULSE
SET_GUNNED

SUM_RETURNS
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(COUNTED)
2		(NEITHER)
3		(SUMMED)


TALLY_RETU
RNS
ACCOUNTED_
FOR
SUM_ENERGY
1
SUM_ENERGY
2
DROP_PULSE
SET_COUNTE
0

TALLY_RETURNS
STRUCTURE LEGEND

NODE ID	ORDINAL VALUE	CONDITIONAL EXPRESSIONS AND/OR COMMENTS
1		(SUMMED)
2		(NEITHER)
3		(COUNTED)

APPENDIX H

TLS REQUIREMENTS

```
ALPHA:  ACKNOWLEDGE.
        BETA: "BEGIN END;".
        FORMS:
              MESSAGE:  ACKNOWLEDGEMENT.
        REFERRED BY:
              R_NET:  CC_RESPONSE.

ALPHA:  ASSIGN_RATE.
        BETA:
        "BEGIN
          SELECT FIRST RECORD FROM STATE_FILE
            SUCH THAT (STATE_ID=IMAGE_ID);
          TRACK_RATE:=STATE_DATA;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' TRACK_RATE=',TRACK_RATE);
        END;",
        INPUTS:
              DATA:  IMAGE_ID
              FILE:  STATE_FILE.
        OUTPUTS:
              DATA:  TRACK_RATE.
        TRACED FROM:
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
              R_NET:  CONTROL_RESOURCES.

ALPHA:  CC_ERROR_PROCESSING.
        BETA:
         "BEGIN
          END;".
        REFERRED BY:
              R_NET:  CC_RESPONSE.

ALPHA:  COMPLETE_SUMMARY.
        BETA:
        "BEGIN
          DATA_RECORD_TYPE:=RADAR_USAGE_REPORT;
          ENGAGEMENT_TIME:=CLOCK_TIME;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' DATA_RECORD_TYPE=',DATA_RECORD_TYPE);
WRITELN(OUTPUT,' ENGAGEMENT_TIME=',ENGAGEMENT_TIME);
         END;".
```

```
      FORMS:
           MESSAGE:  RADAR_USAGE.
      INPUTS:
           DATA:  CLOCK_TIME
           DATA:  RADAR_CLOCK.
      OUTPUTS:
           DATA:  DATA_RECORD_TYPE
           DATA:  ENGAGEMENT_TIME.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           R_NET:  RADAR_SUMMARY.

ALPHA:  CREATE_STATE_FILE.
      BETA:
      "BEGIN
        CREATE STATE_FILE RECORD;
        STATE_ID:=IMAGE_ID;
        STATE_DATA:=STATE;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' STATE_ID=',STATE_ID);
WRITELN(OUTPUT,' STATE_DATA=',STATE_DATA);
       END;".
      INPUTS:
           DATA:  IMAGE_ID
           DATA:  STATE.
      OUTPUTS:
           FILE:  STATE_FILE.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           R_NET:  CONTROL_RESOURCES.

ALPHA:  DROP_LAST.
      BETA: "BEGIN END;".
      GAMMA: "BEGIN END;".
      DESTROYS:
           ENTITY_CLASS:  PULSE.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           R_NET:  CONTROL_RESOURCES.

ALPHA:  DROP_PULSE.
      BETA: "BEGIN END;".
      DESTROYS:
           ENTITY_CLASS:  PULSE.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           SUBNET:  SUM_RETURNS
           SUBNET:  TALLY_RETURNS.
```


BEST AVAILABLE CO

```
ALPHA:  ENGAGEMENT_INITIATION.
        BETA: "BEGIN MODE:=ENGAGED END;".
        OUTPUTS:
             DATA:  MODE.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  CC_RESPONSE.

ALPHA:  FIND_CONFLICT.
        BETA:
        "BEGIN
          DROP_FLAG:=FALSE;
          FOR EACH COMMAND RECORD
           SUCH THAT (START_TIME>TEOF)
           DO
             BEGIN
               DROP_FLAG:=TRUE;
               DESTROY CANDIDATE RECORD;
             END;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' START_TIME=',START_TIME);
WRITELN(OUTPUT,' TEOF=',TEOF);
          ENDFOREACH
        END;",
        DESCRIPTION:
              "COMPARES TRANSMIT RECEIVE WINDOW OF THE CANDIDATE WITH
               THOSE OF THE THEN_CURRENT COMMAND FOR CONFLICT, IF
               A CONFLICT IS FOUND DROP_FLAG IS SET TRUE.".
        INPUTS:
             DATA:  START_TIME
             DATA:  TEOF.
        OUTPUTS:
             DATA:  DROP_FLAG.
        TRACED FROM:
             DECISION:  TRACK_PERFORMANCE_ALLOCATION
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             SUBNET:  FORM_FRAME.

ALPHA:  FORM_T1_T2.
        BETA:
        "BEGIN
          T1_T2_TRANSMIT:=0.0;
          CREATE T1_T2_GATE RECORD;
          T1_T2_GATE_DATA:=0.0;
          RECEIVE_STOP:=TRANSMIT_START;
          T1_T2_XMIT:=0.0;
          CREATE T1_T2_WINDOW RECORD;
          T1_T2_WINDOW_DATA:=0.0;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' T1_T2_TRANSMIT=',T1_T2_TRANSMIT);
WRITELN(OUTPUT,' T1_T2_GATE_DATA=',T1_T2_GATE_DATA);
```

```
WRITELN(OUTPUT,' RECEIVE_STOP=',RECEIVE_STOP);
WRITELN(OUTPUT,' T1_T2_XMIT=',T1_T2_XMIT);
WRITELN(OUTPUT,' T1_T2_WINDOW_DATA=',T1_T2_WINDOW_DATA);
      END;".
      FORMS:
            MESSAGE:  T1_T2_COMMAND.
      INPUTS:
            DATA:  TRANSMIT_START.
      OUTPUTS:
            DATA:  RECEIVE_STOP
            DATA:  T1_T2_TRANSMIT
            DATA:  T1_T2_XMIT
            FILE:  T1_T2_GATE
            FILE:  T1_T2_WINDOW.
      SETS:
            ENTITY_TYPE:  T1_T2_PULSE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  XMIT_R.

ALPHA:  FORM_T3.
      BETA:
      "BEGIN
        T3_TRANSMIT:=0.0;
        CREATE T3_GATE RECORD;
        T3_GATE_DATA:=0.0;
        T3_XMIT:=0.0;
        RECEIVE_STOP:=TRANSMIT_START;
        CREATE T3_WINDOW RECORD;
        T3_WINDOW_DATA:=0.0;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' T3_TRANSMIT=',T3_TRANSMIT);
WRITELN(OUTPUT,' T3_GATE_DATA=',T3_GATE_DATA);
WRITELN(OUTPUT,' T3_XMIT=',T3_XMIT);
WRITELN(OUTPUT,' RECEIVE_STOP=',RECEIVE_STOP);
WRITELN(OUTPUT,' T3_WINDOW_DATA=',T3_WINDOW_DATA);
      END;".
      FORMS:
            MESSAGE:  T3_COMMAND.
      INPUTS:
            DATA:  TRANSMIT_START.
      OUTPUTS:
            DATA:  RECEIVE_STOP
            DATA:  T3_TRANSMIT
            DATA:  T3_XMIT
            FILE:  T3_GATE
            FILE:  T3_WINDOW.
      SETS:
            ENTITY_TYPE:  T3_PULSE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
```

```
        REFERRED BY:
             R_NET:  XMIT_R.

   ALPHA:  FORM_UPDATE.
        BETA: "BEGIN DATA_RECORD_TYPE:=STATE_UPDATE_REPORT END;".
        FORMS:
             MESSAGE:  STATE_UPDATE.
        OUTPUTS:
             DATA:  DATA_RECORD_TYPE.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

   ALPHA:  GHOST_DETERMINATION.
        BETA:
        "BEGIN
          GHOST_IMAGE:=(RADAR_MEASUREMENT>=10.0);
   WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
   WRITELN(OUTPUT,' RADAR_MEASUREMENT=',RADAR_MEASUREMENT);
   WRITELN(OUTPUT,' GHOST_IMAGE=',GHOST_IMAGE);
         END;".
        INPUTS:
             DATA:  RADAR_MEASUREMENT.
        OUTPUTS:
             DATA:  GHOST_IMAGE.
        TRACED FROM:
             DECISION:  TRACK_PERFORMANCE_ALLOCATION
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

   ALPHA:  GHOST_TERMINATION.
        BETA:
     "BEGIN CREATE TERMINATOR RECORD;
      HO_ID:=IMAGE_ID;
      REASON_FOR_DROP:=GHOST;
      DROP_REASON:=GHOST;
      TIME_OF_DROP:=CLOCK_TIME;
DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT;
      DROP_TIME:=CLOCK_TIME END;".
        FORMS:
             MESSAGE:  TRACK_TERMINATION.
        INPUTS:
             DATA:  CLOCK_TIME
             DATA:  IMAGE_ID.
        OUTPUTS:
             DATA:  DATA_RECORD_TYPE
```

```
            DATA:  HO_ID
            DATA:  REASON_FOR_DROP
            DATA:  TIME_OF_DROP
            FILE:  TERMINATOR.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  RESPONSE_TO_RADAR.

 ALPHA:  INITIALIZE_SKED_R.
      BETA:
      "BEGIN
        TEMP:=CLOCK_TIME+FRAME_RATE;
        IST:=CLOCK_TIME;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' FRAME_RATE=',FRAME_RATE);
WRITELN(OUTPUT,' TEMP=',TEMP);
WRITELN(OUTPUT,' IST=',IST);
       END;",
      DESCRIPTION:
      "COMPUTES THE TIME OF THE END OF THE CURRENT FRAME.",
      INPUTS:
            DATA:  CLOCK_TIME
            DATA:  FRAME_RATE.
      OUTPUTS:
            DATA:  IST
            DATA:  TEMP.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  SKED_R.

 ALPHA:  LOW_ELEVATION_DETERMINATION.
      BETA:
      "BEGIN
        LOW_ELEVATION:=(CLOCK_TIME-ENTRY_TIME>ELEVATION_LIMIT);
        CURRENT_STATE:=CURRENT_STATE;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' LOW_ELEVATION=',LOW_ELEVATION);
WRITELN(OUTPUT,' ENTRY_TIME=',ENTRY_TIME);
```

```
WRITELN(OUTPUT,' ELEVATION_LIMIT=',ELEVATION_LIMIT);
      END;",
      INPUTS:
            DATA:  CURRENT_STATE
            DATA:  ELEVATION_LIMIT
            DATA:  ENTRY_TIME.
      OUTPUTS:
            DATA:  LOW_ELEVATION.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  RESPONSE_TO_RADAR.

ALPHA:  LOW_TERMINATION.
      BETA:
  "BEGIN CREATE TERMINATOR RECORD;
 HO_ID:=IMAGE_ID;
 REASON_FOR_DROP:=LOW;
 DROP_REASON:=LOW;
 TIME_OF_DROP:=CLOCK_TIME;
 DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT;
 DROP_TIME:=CLOCK_TIME END;",
      FORMS:
            MESSAGE:  TRACK_TERMINATION.
      INPUTS:
            DATA:  CLOCK_TIME
            DATA:  IMAGE_ID.
      OUTPUTS:
            DATA:  DATA_RECORD_TYPE
            DATA:  HO_ID
            DATA:  REASON_FOR_DROP
            DATA:  TIME_OF_DROP
            FILE:  TERMINATOR.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  RESPONSE_TO_RADAR.

ALPHA:  MAKE_ALLOCATION.
      BETA:
      "VAR J,V,DELTA_TIME:REAL;
       BEGIN
        DELTA_TIME:=CLOCK_TIME-LAST_ALLOCATE;
  WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
  WRITELN(OUTPUT,' DELTA_TIME=',DELTA_TIME);
WRITELN(OUTPUT,' LAST_ALLOCATE=',LAST_ALLOCATE);
```

```
  WRITELN(OUTPUT,' ENERGY=',ENERGY);
          LAST_ALLOCATE:=CLOCK_TIME;
WRITELN(OUTPUT,' LAST_ALLOCATE=',LAST_ALLOCATE);
          IF ENERGY > 0.0 THEN
            BEGIN
              ENERGY:=ENERGY/DELTA_TIME;
WRITELN(OUTPUT,' ENERGY=',ENERGY);
              J:=0.0;
              FOR EACH STATE_FILE RECORD
                DO
                J:=J+1.0;
WRITELN(OUTPUT,' J=',J);
              ENDFOREACH;
              IF J > 0.0 THEN
                BEGIN
                  V:=ENERGY/J;
WRITELN(OUTPUT,' V=',V);
                END
              ELSE
            END
          ELSE
            V:=TRACK_RATE;
WRITELN(OUTPUT,' V=',V);
          FOR EACH STATE_FILE RECORD
            DO
            STATE_DATA:=V;
WRITELN(OUTPUT,' STATE_DATA=',STATE_DATA);
          ENDFOREACH;
  WRITELN(OUTPUT,' STATE_DATA=',STATE_DATA);
        END;".
        INPUTS:
             DATA:  CLOCK_TIME
             DATA:  ENERGY
             DATA:  LAST_ALLOCATE
             FILE:  STATE_FILE.
        OUTPUTS:
             DATA:  ENERGY
             DATA:  LAST_ALLOCATE
             DATA:  STATE_DATA.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  CONTROL_RESOURCES.

   ALPHA:  MAKE_COMMAND.
        BETA:
        "BEGIN
          CREATE COMMAND RECORD;
          COMMAND_IMAGE_ID:=CANDIDATE_IMAGE_ID;
          COMMAND_WAVEFORM:=CANDIDATE_WAVEFORM;
          COMMAND_ENERGY:=CANDIDATE_ENERGY;
          START_TIME:=IST;
          IST:=IST+DELTAT;
  WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
  WRITELN(OUTPUT,' COMMAND_IMAGE_ID=',COMMAND_IMAGE_ID);
```

```
WRITELN(OUTPUT,' COMMAND_WAVEFORM=',COMMAND_WAVEFORM);
WRITELN(OUTPUT,' COMMAND_ENERGY=',COMMAND_ENERGY);
WRITELN(OUTPUT,' START_TIME=',START_TIME);
WRITELN(OUTPUT,' IST=',IST);
      END;",
      INPUTS:
            DATA:  CANDIDATE_ENERGY
            DATA:  CANDIDATE_IMAGE_ID
            DATA:  CANDIDATE_WAVEFORM
            DATA:  DELTAT
            DATA:  TST.
      OUTPUTS:
            DATA:  COMMAND_ENERGY
            DATA:  COMMAND_IMAGE_ID
            DATA:  COMMAND_WAVEFORM
            DATA:  IST
            DATA:  START_TIME.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            SUBNET:  FORM_FRAME.

ALPHA:  PICK_CANDIDATES.
      BETA:
      "BEGIN
        CREATE CANDIDATE RECORD;
        CANDIDATE_IMAGE_ID:=IMAGE_ID;
        IF WAVEFORM=T1 THEN
           PRIORITY:=1.0
        ELSE
        IF WAVEFORM=T2 THEN
           PRIORITY:=2.0
        ELSE
           PRIORITY:=3.0;
        CANDIDATE_WAVEFORM:=WAVEFORM;
        SELECT FIRST RECORD FROM WAVEFORM_TABLE
          SUCH THAT WF_NAME=WAVEFORM;
        IF NOT RECORD_FOUND THEN
          CANDIDATE_ENERGY:=0.0
        ELSE
          CANDIDATE_ENERGY:=WF_CHARACTERISTICS;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' LAST_PULSE=',LAST_PULSE);
WRITELN(OUTPUT,' TEMF=',TEMF);
WRITELN(OUTPUT,' TRACK_RATE=',TRACK_RATE);
WRITELN(OUTPUT,' CANDIDATE_IMAGE_ID=',CANDIDATE_IMAGE_ID);
WRITELN(OUTPUT,' WAVEFORM=',WAVEFORM);
WRITELN(OUTPUT,' PRIORITY=',PRIORITY);
WRITELN(OUTPUT,' CANDIDATE_WAVEFORM=',CANDIDATE_WAVEFORM);
WRITELN(OUTPUT,' CANDIDATE_ENERGY=',CANDIDATE_ENERGY);
       END;",
      DESCRIPTION:
            "EACH IMAGE_IN_TRACK WHICH MIGHT GENERATE A MESSAGE THIS
             FRAME HAS ITS TRANSMIT AND RECEIVE START AND STOP TIMES
             EXTRACTED, ITS ENERGY DEMAND DETERMINED AND ITS
```

```
                PRIORITY ESTABLISHED.",
        ENTERED_BY: "M.RICHTER",
        INPUTS:
             DATA:  IMAGE_ID
             DATA:  WAVEFORM
             (*INSTANCES OF ENTITY_TYPE IMAGE_IN_TRACK*)
             FILE:  WAVEFORM_TABLE,
        OUTPUTS:
             DATA:  CANDIDATE_ENERGY
             DATA:  CANDIDATE_IMAGE_ID
             DATA:  CANDIDATE_WAVEFORM
             DATA:  PRIORITY.
        TRACED FROM:
             DECISION:  SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  SHED_R,

   ALPHA:  PICK_COMMAND.
        BETA:
        "BEGIN
          TRANSMIT_START:=START_TIME;
          RADAR_TYPE:=COMMAND_WAVEFORM;
          RR_ORDER_IDC:=RR_ORDER_IDC+1;
          RR_ORDER_ID:=RR_ORDER_IDC;
          PULSE_TYPE:=COMMAND_WAVEFORM;
          TARGET_ID:=COMMAND_IMAGE_ID;
          PULSE_ID:=RR_ORDER_IDC;
          XMIT_START:=START_TIME;
          DESTROY COMMAND RECORD;
   WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
   WRITELN(OUTPUT,' TRANSMIT_START=',TRANSMIT_START);
   WRITELN(OUTPUT,' RADAR_TYPE=',RADAR_TYPE);
   WRITELN(OUTPUT,' RR_ORDER_IDC=',RR_ORDER_IDC);
   WRITELN(OUTPUT,' RR_ORDER_ID=',RR_ORDER_ID);
   WRITELN(OUTPUT,' PULSE_TYPE=',PULSE_TYPE);
   WRITELN(OUTPUT,' TARGET_ID=',TARGET_ID);
   WRITELN(OUTPUT,' PULSE_ID=',PULSE_ID);
   WRITELN(OUTPUT,' XMIT_START=',XMIT_START);
          END;",
        DESCRIPTION: "PICK_COMMAND SELECTS NEXT COMMAND.",
        CREATES:
             ENTITY_CLASS:  PULSE.
        INPUTS:
             DATA:  RR_ORDER_IDC
             DATA:  START_TIME
             DATA:  XMIT_START
             FILE:  COMMAND.
        OUTPUTS:
             DATA:  PULSE_ID
             DATA:  PULSE_TYPE
             DATA:  RADAR_TYPE
             DATA:  RR_ORDER_ID
             DATA:  RR_ORDER_IDC
             DATA:  TARGET_ID
             DATA:  TRANSMIT_START
             DATA:  XMIT_START.
```

```
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           R_NET:  XMIT_R.

 ALPHA:  REDUN_DETERMINATION.
      BETA:
      "BEGIN
        REDUNDANT_IMAGE:=((REDUNDANT_IMAGE) OR
                          (STATE=CURRENT_STATE));
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' CURRENT_STATE=',CURRENT_STATE);
WRITELN(OUTPUT,' STATE=',STATE);
WRITELN(OUTPUT,' REDUNDANT_IMAGE=',REDUNDANT_IMAGE);
       END;",
      INPUTS:
           DATA:  CURRENT_STATE
           DATA:  STATE.
      OUTPUTS:
           DATA:  REDUNDANT_IMAGE.
      TRACED FROM:
           DECISION:  TRACK_PERFORMANCE_ALLOCATION
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           R_NET:  RESPONSE_TO_RADAR.

 ALPHA:  REDUN_TERMINATION.
      BETA:
      "BEGIN
        CREATE TERMINATOR RECORD;
        HO_ID:=IMAGE_ID;
        TIME_OF_DROP:=CLOCK_TIME;
        DROP_TIME:=CLOCK_TIME;
        REASON_FOR_DROP:=REDUNDANT;
        DROP_REASON:=REDUNDANT;
        DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' HO_ID=',HO_ID);
WRITELN(OUTPUT,' TIME_OF_DROP=',TIME_OF_DROP);
WRITELN(OUTPUT,' DROP_TIME=',DROP_TIME);
WRITELN(OUTPUT,' REASON_FOR_DROP=',REASON_FOR_DROP);
WRITELN(OUTPUT,' DROP_REASON=',DROP_REASON);
WRITELN(OUTPUT,' DATA_RECORD_TYPE=',DATA_RECORD_TYPE);
       END;",
      FORMS:
           MESSAGE:  TRACK_TERMINATION.
```

```
        INPUTS:
             DATA:  CLOCK_TIME
             DATA:  IMAGE_ID.
        OUTPUTS:
             DATA:  DATA_RECORD_TYPE
             DATA:  HO_ID
             DATA:  REASON_FOR_DROP
             DATA:  TIME_OF_DROP
             FILE:  TERMINATOR.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

   ALPHA:  REMEMBER_STOP.
        BETA:
        "BEGIN
          STOP_TIME:=RECEIVE_STOP;
  WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
  WRITELN(OUTPUT,' STOP_TIME=',STOP_TIME);
         END;".
        INPUTS:
             DATA:  RECEIVE_STOP.
        OUTPUTS:
             DATA:  STOP_TIME.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

   ALPHA:  RP_ERROR_PROCESSING.
        BETA: "BEGIN END;".
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

   ALPHA:  SELECT_COMMAND.
        BETA: "BEGIN SELECT FIRST RECORD FROM COMMAND END;".
        INPUTS:
             FILE:  COMMAND.
        OUTPUTS:
             DATA:  RECORD_FOUND.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  XMIT_R.

   ALPHA:  SET_COUNTED.
        BETA:
        "BEGIN
```

```
        ACCOUNTED_FOR:=COUNTED;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' ACCOUNTED_FOR=',ACCOUNTED_FOR);
       END;".
      OUTPUTS:
           DATA:  ACCOUNTED_FOR.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           SUBNET:  TALLY_RETURNS.

 ALPHA:  SET_DROP.
      BETA: "BEGIN END;".
      DESCRIPTION: "SETS TYPE FOR IMAGE TO BE DROPPED".
      SETS:
           ENTITY_TYPE:  DROPPED_IMAGE.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           SUBNET:  RECORD_DROP.

 ALPHA:  SET_LAST.
      BETA:
      "BEGIN
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' LAST_PULSE=',LAST_PULSE);
WRITELN(OUTPUT,' START_TIME=',START_TIME);
        LAST_PULSE:=START_TIME;
        DESTROY CANDIDATE RECORD;
WRITELN(OUTPUT,' LAST_PULSE=',LAST_PULSE);
       END;",
      INPUTS:
           DATA:  LAST_PULSE
           DATA:  START_TIME.
      OUTPUTS:
           DATA:  LAST_PULSE.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           SUBNET:  FORM_FRAME.

 ALPHA:  SET_LAST.
      BETA: "BEGIN END;".
      DESCRIPTION: "RECORDS LOSS OF PULSE WHEN OVERDUE.",
      INPUTS:
           DATA:  T1_T2_WINDOW_DATA
           DATA:  T1_T2_XMIT
           DATA:  T3_WINDOW_DATA
           DATA:  T3_XMIT.
      SETS:
           ENTITY_TYPE:  LAST_PULSE.
      REFERRED BY:
           SUBNET:  MISSING_RETURNS.

 ALPHA:  SET_PULSE.
```

```
      BETA: "BEGIN END;".
      DESCRIPTION: "RECORDS VALID, CORRELATED RETURN.",
      SETS:
           ENTITY_TYPE:  RETURNED_PULSE.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           SUBNET:  EXTRACT_MEASUREMENT.

 ALPHA:  SET_SUMMED.
      BETA:
      "BEGIN
        ACCOUNTED_FOR:=SUMMED;
WRITELN (OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN (OUTPUT,' ACCOUNTED_FOR=',ACCOUNTED_FOR);
       END;".
      OUTPUTS:
           DATA:  ACCOUNTED_FOR.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           SUBNET:  SUM_RETURNS.

 ALPHA:  STARTER.
      ARTIFICIALITY: ARTIFICIAL.
      BETA:
      "BEGIN CREATE WAVEFORM_TABLE RECORD;
             WF_NAME:=T1;
             WF_CHARACTERISTICS:=1.0;
             CREATE WAVEFORM_TABLE RECORD;
             WF_NAME:=T2;
             WF_CHARACTERISTICS:=2.0;
             CREATE WAVEFORM_TABLE RECORD;
             WF_NAME:=T3;
             WF_CHARACTERISTICS:=3.0
       END;".
      DESCRIPTION: "THIS ELEMENT INITIALIZES WAVEFORM_TABLE".
      OUTPUTS:
           FILE:  WAVEFORM_TABLE.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           R_NET:  CC_RESPONSE.

 ALPHA:  SUM_ENERGY.
      BETA:
      "BEGIN
        SELECT FIRST RECORD FROM WAVEFORM_TABLE
          SUCH THAT (WF_NAME=PULSE_TYPE);
          IF RECORD_FOUND THEN
             ENERGY:=ENERGY+WF_CHARACTERISTICS;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' PULSE_TYPE=',PULSE_TYPE);
```

```
WRITELN(OUTPUT,' ENERGY=',ENERGY);
        END;".
        INPUTS:
             DATA:  ENERGY
             DATA:  PULSE_TYPE
             FILE:  WAVEFORM_TABLE.
        OUTPUTS:
             DATA:  ENERGY.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             SUBNET:  TALLY_RETURNS.

   ALPHA:  SUM_USAGE.
        BETA:
        "BEGIN
          SELECT FIRST RECORD FROM WAVEFORM_TABLE
            SUCH THAT (WF_NAME=PULSE_TYPE);
          RESOURCES:=RESOURCES+WF_CHARACTERISTICS;
   WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
   WRITELN(OUTPUT,' PULSE_TYPE=',PULSE_TYPE);
   WRITELN(OUTPUT,' RESOURCES=',RESOURCES);
         END;",
        OUTPUTS:
             DATA:  RESOURCES.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             SUBNET:  SUM_RETURNS.

   ALPHA:  TERM_ENGAGEMENT.
        BETA: "BEGIN MODE:=STANDBY END;".
        OUTPUTS:
             DATA:  MODE.
        REFERRED BY:
             R_NET:  CC_RESPONSE.

   ALPHA:  TERM_TRACK.
        BETA:
      "BEGIN CREATE TERMINATOR RECORD; DROP_TIME:=CLOCK_TIME;
DROP_REASON:=CC_COMMAND_TO_DROP;   REASON_FOR_DROP:=DROP_REASON;
TIME_OF_DROP:=DROP_TIME;
DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT END;".
        FORMS:
             MESSAGE:  TRACK_TERMINATION.
        INPUTS:
             DATA:  CLOCK_TIME
             DATA:  DROP_REASON
             DATA:  DROP_TIME.
        OUTPUTS:
             DATA:  DATA_RECORD_TYPE
             DATA:  REASON_FOR_DROP
             DATA:  TIME_OF_DROP
             FILE:  TERMINATOR.
```

```
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  CC_RESPONSE.

   ALPHA:  TRACK_INITIATE.
        BETA:
        "BEGIN
          TIME_OF_INITIATION:=CLOCK_TIME;
          IMAGE_ID:=HQ_ID;
          STATE:=INITIAL_STATE;
          COVARIANCE:=INITIAL_COVARIANCE;
          TRACK_RATE:=20.0;
          WAVEFORM:=T1;
          DATA_RECORD_TYPE:=TRACK_INITIATION_REPORT;
          ENTRY_TIME:=CLOCK_TIME;
  WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
  WRITELN(OUTPUT,' TIME_OF_INITIATION=',TIME_OF_INITIATION);
  WRITELN(OUTPUT,' IMAGE_ID=',IMAGE_ID);
  WRITELN(OUTPUT,' STATE=',STATE);
  WRITELN(OUTPUT,' COVARIANCE=',COVARIANCE);
  WRITELN(OUTPUT,' TRACK_RATE=',TRACK_RATE);
  WRITELN(OUTPUT,' WAVEFORM=',WAVEFORM);
  WRITELN(OUTPUT,' DATA_RECORD_TYPE=',DATA_RECORD_TYPE);
  WRITELN(OUTPUT,' ENTRY_TIME=',ENTRY_TIME);
          END;".
        CREATES:
             ENTITY_CLASS:  IMAGE.
        FORMS:
             MESSAGE:  TRACK_INITIATION.
        INPUTS:
             DATA:  CLOCK_TIME
             DATA:  HQ_ID
             DATA:  INITIAL_COVARIANCE
             DATA:  INITIAL_STATE.
        OUTPUTS:
             DATA:  COVARIANCE
             DATA:  DATA_RECORD_TYPE
             DATA:  ENTRY_TIME
             DATA:  IMAGE_ID
             DATA:  STATE
             DATA:  TIME_OF_INITIATION
             DATA:  TRACK_RATE
             DATA:  WAVEFORM.
        SETS:
             ENTITY_TYPE:  IMAGE_IN_TRACK.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_OPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  CC_RESPONSE.
```

```
 ALPHA:  T1_T2_MEASUREMENT_EXTRACTION.
       BETA:
       "BEGIN
         VALID_RETURN:=TRUE;
         RADAR_MEASUREMENT:=T1_T2_RECEIVE;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' VALID_RETURN=',VALID_RETURN);
WRITELN(OUTPUT,' RADAR_MEASUREMENT=',RADAR_MEASUREMENT);
        END;",
       INPUTS:
            DATA:  T1_T2_RECEIVE
            FILE:  T1_T2_DATA.
       OUTPUTS:
            DATA:  RADAR_MEASUREMENT
            DATA:  VALID_RETURN.
       TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS.
       REFERRED BY:
            SUBNET:  EXTRACT_MEASUREMENT.

 ALPHA:  T3_MEASUREMENT_EXTRACTION.
       BETA:
       "BEGIN
         VALID_RETURN:=ODD(TRUNC(T3_RECEIVE+0.1));
         RADAR_MEASUREMENT:=T3_RECEIVE;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' T3_RECEIVE=',T3_RECEIVE);
WRITELN(OUTPUT,' VALID_RETURN=',VALID_RETURN);
WRITELN(OUTPUT,' RADAR_MEASUREMENT=',RADAR_MEASUREMENT);
        END;",
       INPUTS:
            DATA:  T3_RECEIVE
            FILE:  T3_DATA.
       OUTPUTS:
            DATA:  RADAR_MEASUREMENT
            DATA:  VALID_RETURN.
       TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS.
       REFERRED BY:
            SUBNET:  EXTRACT_MEASUREMENT.

 ALPHA:  UPDATE_RADAR_CLOCK.
       BETA:
        "BEGIN
            RADAR_CLOCK:=RADAR_CLOCK_TIME
         END;".
       INPUTS:
            DATA:  RADAR_CLOCK_TIME.
       OUTPUTS:
            DATA:  RADAR_CLOCK.
       REFERRED BY:
```

```
            R_NET:  RADAR_TIMING.

ALPHA:  UPDATE_STATE.
      BETA:
      "BEGIN
        IF WAVEFORM=T1 THEN
           WAVEFORM:=T2
        ELSE
        IF WAVEFORM=T2 THEN
           WAVEFORM:=T3
        ELSE
           WAVEFORM:=T1;
        MS_ID:=IMAGE_ID;
        STATE:=RADAR_MEASUREMENT;
        CURRENT_STATE:=STATE;
        COVARIANCE:=CLOCK_TIME;
WRITELN(OUTPUT,' CLOCK_TIME=',CLOCK_TIME);
WRITELN(OUTPUT,' WAVEFORM=',WAVEFORM);
WRITELN(OUTPUT,' MS_ID=',MS_ID);
WRITELN(OUTPUT,' STATE=',STATE);
WRITELN(OUTPUT,' CURRENT_STATE=',CURRENT_STATE);
WRITELN(OUTPUT,' COVARIANCE=',COVARIANCE);
       END;".
      INPUTS:
            DATA:  CLOCK_TIME
            DATA:  COVARIANCE
            DATA:  IMAGE_ID
            DATA:  RADAR_MEASUREMENT
            DATA:  WAVEFORM.
      OUTPUTS:
            DATA:  COVARIANCE
            DATA:  CURRENT_STATE
            DATA:  MS_ID
            DATA:  STATE
            DATA:  WAVEFORM.
      TRACED FROM:
            DECISION:  TRACK_PERFORMANCE_ALLOCATION
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  RESPONSE_TO_RADAR.

ALPHA:  VALIDATE_HEADER.
      BETA:
      "BEGIN
        COMMAND_ID:=COMMAND_ID;
       END;".
      INPUTS:
            DATA:  COMMAND_ID.
      OUTPUTS:
            DATA:  COMMAND_ID.
      TRACED FROM:
```

```
            ORIGINATING_REQUIREMENT:
                TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
            R_NET:  CC_RESPONSE.

DATA:  ACCEPTANCE_THRESHOLD.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
            DATA:  T1_T2_GATE_DATA
            DATA:  T3_GATE_DATA.

DATA:  ACCOUNTED_FOR.
        INITIAL_VALUE: NEITHER.
        LOCALITY: GLOBAL.
        RANGE: "NEITHER,COUNTED,SUMMED".
        TYPE: ENUMERATION.
        USE: BOTH.
        ASSOCIATED WITH:
            ENTITY_TYPE:  LAST_PULSE
            ENTITY_TYPE:  RETURNED_PULSE.
        OUTPUT FROM:
            ALPHA:  SET_COUNTED
            ALPHA:  SET_SUMMED.
        REFERRED BY:
            SUBNET:  SUM_RETURNS
            SUBNET:  TALLY_RETURNS.

DATA:  ALPHA_ERROR.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
            DATA:  T1_T2_RECEIVE
            DATA:  T3_RECEIVE.

DATA:  ALPHA_PHASE_TAPER.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
            DATA:  T1_T2_TRANSMIT
            DATA:  T3_TRANSMIT.

DATA:  AVERAGE_SIGNAL_POWER.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
            DATA:  WAVE_ARRAY.

DATA:  BETA_ERROR.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
            DATA:  T1_T2_RECEIVE
```

```
            DATA:  T3_RECEIVE.

DATA:  BETA_PHASE_TAPER.
       LOCALITY: LOCAL.
       TYPE: REAL.
       USE: GAMMA.
       INCLUDED IN:
            DATA:  T1_T2_TRANSMIT
            DATA:  T3_TRANSMIT.

DATA:  CANDIDATE_ENERGY.
       LOCALITY: GLOBAL.
       TYPE: REAL.
       USE: BOTH.
       CONTAINED IN:
            FILE:  CANDIDATE.
       INPUT TO:
            ALPHA:  MAKE_COMMAND.
       OUTPUT FROM:
            ALPHA:  PICK_CANDIDATES.

DATA:  CANDIDATE_IMAGE_ID.
       LOCALITY: GLOBAL.
       TYPE: INTEGER.
       USE: BOTH.
       CONTAINED IN:
            FILE:  CANDIDATE.
       INPUT TO:
            ALPHA:  MAKE_COMMAND.
       OUTPUT FROM:
            ALPHA:  PICK_CANDIDATES.
       REFERRED BY:
            SUBSET:  FORM_FRAME.

DATA:  CANDIDATE_WAVEFORM.
       LOCALITY: GLOBAL.
       RANGE: "T1,T2,T3".
       TYPE: ENUMERATION.
       USE: BOTH.
       CONTAINED IN:
            FILE:  CANDIDATE.
       INPUT TO:
            ALPHA:  MAKE_COMMAND.
       OUTPUT FROM:
            ALPHA:  PICK_CANDIDATES.

DATA:  CLOCK_TIME
                (* A PREDEFINED DATA ITEM WHICH INCREMENTS AT THE
                   SAME RATE AS ENGAGEMENT TIME.  EXCEPT FOR ITS
                   INITIAL_VALUE WHICH IS ARBITRARY, CLOCK_TIME MAY
                   BE REGARDED AS ENGAGEMENT TIME.  IT HAS NO CLOCK
                   ERROR, *).
       LOCALITY: GLOBAL.
       TYPE: REAL.
       UNITS: SECONDS.
       USE: BOTH.
       INPUT TO:
            ALPHA:  COMPLETE_SUMMARY
```

```
            ALPHA:  GHOST_TERMINATION
            ALPHA:  INITIALIZE_SKED_R
            ALPHA:  LOW_TERMINATION
            ALPHA:  MAKE_ALLOCATION
            ALPHA:  REDUN_TERMINATION
            ALPHA:  TERM_TRACK
            ALPHA:  TRACK_INITIATE
            ALPHA:  UPDATE_STATE.

DATA:  COMMAND_ENERGY.
      LOCALITY: GLOBAL.
      TYPE: REAL.
      USE: BOTH.
      CONTAINED IN:
            FILE:  COMMAND.
      OUTPUT FROM:
            ALPHA:  MAKE_COMMAND.

DATA:  COMMAND_ID.
      LOCALITY: LOCAL.
      RANGE:
      "HANDOVER_IMAGE,DROP_TRACK,INITIATE_ENGAGEMENT_MODE,
       TERMINATE_ENGAGEMENT_MODE,CC_MESSAGE_ERROR".
      TYPE: ENUMERATION.
      USE: BOTH.
      MAKES:
            MESSAGE:  ACKNOWLEDGEMENT
            MESSAGE:  HANDOVER
            MESSAGE:  MODE_CHANGE
            MESSAGE:  TERMINATION.
      INPUT TO:
            ALPHA:  VALIDATE_HEADER.
      OUTPUT FROM:
            ALPHA:  VALIDATE_HEADER.
      REFERRED BY:
            R_NET:  CC_RESPONSE.

DATA:  COMMAND_IMAGE_ID.
      LOCALITY: GLOBAL.
      TYPE: INTEGER.
      USE: BOTH.
      CONTAINED IN:
            FILE:  COMMAND.
      OUTPUT FROM:
            ALPHA:  MAKE_COMMAND.

DATA:  COMMAND_WAVEFORM.
      LOCALITY: GLOBAL.
      RANGE: "T1,T2,T3".
      TYPE: ENUMERATION.
      USE: BOTH.
      CONTAINED IN:
            FILE:  COMMAND.
      OUTPUT FROM:
            ALPHA:  MAKE_COMMAND.

DATA:  COVARIANCE.
      LOCALITY: GLOBAL.
```

```
      TYPE: REAL.
      USE: BETA.
      ASSOCIATED WITH:
           ENTITY_TYPE:  IMAGE_IN_TRACK.
      INPUT TO:
           ALPHA:  UPDATE_STATE.
      OUTPUT FROM:
           ALPHA:  TRACK_INITIATE
           ALPHA:  UPDATE_STATE.

DATA:  CURRENT_STATE.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BETA.
      MAKES:
           MESSAGE:  STATE_UPDATE.
      INPUT TO:
           ALPHA:  LOW_ELEVATION_DETERMINATION
           ALPHA:  REDUN_DETERMINATION.
      OUTPUT FROM:
           ALPHA:  UPDATE_STATE.

DATA:  DATA_RECORD_TYPE.
      LOCALITY: LOCAL.
      RANGE:
       "RADAR_USAGE_REPORT,STATE_UPDATE_REPORT,
        TRACK_TERMINATION_REPORT,TRACK_INITIATION_REPORT".
      TYPE: ENUMERATION.
      USE: BOTH.
      MAKES:
           MESSAGE:  RADAR_USAGE
           MESSAGE:  STATE_UPDATE
           MESSAGE:  TRACK_INITIATION
           MESSAGE:  TRACK_TERMINATION.
      OUTPUT FROM:
           ALPHA:  COMPLETE_SUMMARY
           ALPHA:  FORM_UPDATE
           ALPHA:  GHOST_TERMINATION
           ALPHA:  LOW_TERMINATION
           ALPHA:  REDUN_TERMINATION
           ALPHA:  TERM_TRACK
           ALPHA:  TRACK_INITIATE.

DATA:  DELTAT.
      DESCRIPTION: "MINIMUM PULSE SPACING FOR BEAM SWITCHING.".
      INITIAL_VALUE: 3.0E-6.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BOTH.
      INPUT TO:
           ALPHA:  MAKE_COMMAND.

DATA:  DROP_FLAG.
      LOCALITY: LOCAL.
      TYPE: BOOLEAN.
      USE: BOTH.
      OUTPUT FROM:
           ALPHA:  FIND_CONFLICT.
```

```
        REFERRED BY:
            SUBNET:  FORM_FRAME.

DATA:  DROP_REASON.
        LOCALITY: GLOBAL.
        RANGE: "GHOST,REDUNDANT,LOW,CC_COMMAND_TO_DROP".
        TYPE: ENUMERATION.
        USE: BOTH.
        CONTAINED IN:
            FILE:  TERMINATOR.
        INPUT TO:
            ALPHA:  TERM_TRACK.
        TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS.

DATA:  DROP_TIME.
        LOCALITY: GLOBAL.
        TYPE: REAL.
        USE: BOTH.
        CONTAINED IN:
            FILE:  TERMINATOR.
        INPUT TO:
            ALPHA:  TERM_TRACK.

DATA:  ELEVATION_LIMIT.
        INITIAL_VALUE: 15.0.
        LOCALITY: GLOBAL.
        TYPE: REAL.
        USE: BOTH.
        INPUT TO:
            ALPHA:  LOW_ELEVATION_DETERMINATION.

DATA:  ENERGY.
        INITIAL_VALUE: 0.0.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: BETA.
        INPUT TO:
            ALPHA:  MAKE_ALLOCATION
            ALPHA:  SUM_ENERGY.
        OUTPUT FROM:
            ALPHA:  MAKE_ALLOCATION
            ALPHA:  SUM_ENERGY.

DATA:  ENGAGEMENT_TIME.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: BOTH.
        MAKES:
            MESSAGE:  RADAR_USAGE.
        OUTPUT FROM:
```

```
            ALPHA:  COMPLETE_SUMMARY.

DATA:  ENTRY_TIME.
       LOCALITY: GLOBAL.
       TYPE: REAL.
       USE: BOTH.
       ASSOCIATED WITH:
            ENTITY_CLASS:  IMAGE.
       INPUT TO:
            ALPHA:  LOW_ELEVATION_DETERMINATION.
       OUTPUT FROM:
            ALPHA:  TRACK_INITIATE.
       TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_OPSPR_PARAGRAPH_3_2_2_F_FUNCTIONAL_REQUIREMENTS.

DATA:  FOUND
                  (* A PREDEFINED DATA ITEM WHICH IS SET TO EITHER
                     TRUE OR FALSE AFTER EACH SELECT ON AN ENTITY_CLASS
                     OR ENTITY_TYPE.  FOUND IS SET TO TRUE IF AN
                     INSTANCE SATISFYING THE SLECTION CRITERION IS
                     LOCATED.  OTHERWISE, FOUND IS ASSIGNED THE
                     VALUE FALSE. *).
       INITIAL_VALUE: FALSE.
       LOCALITY: LOCAL.
       TYPE: BOOLEAN.
       USE: BOTH.
       REFERRED BY:
            R_NET:  CC_RESPONSE
            R_NET:  RESPONSE_TO_RADAR.

DATA:  FRAME_RATE.
       INITIAL_VALUE: 0.01.
       LOCALITY: GLOBAL.
       TYPE: REAL.
       USE: BOTH.
       DELAYS:
            EVENT:  SCHEDULE.
       INPUT TO:
            ALPHA:  INITIALIZE_SKED_R.

DATA:  GATE_LENGTH.
       LOCALITY: LOCAL.
       TYPE: REAL.
       USE: GAMMA.
       INCLUDED IN:
            DATA:  T1_T2_GATE_DATA
            DATA:  T3_GATE_DATA.

DATA:  GATE_START_TIME.
       LOCALITY: LOCAL.
       TYPE: REAL.
       USE: GAMMA.
       INCLUDED IN:
            DATA:  T1_T2_GATE_DATA
            DATA:  T3_GATE_DATA.

DATA:  GHOST_IMAGE.
```

```
        LOCALITY: LOCAL.
        TYPE: BOOLEAN.
        USE: BOTH.
        OUTPUT FROM:
             ALPHA:  GHOST_DETERMINATION.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

DATA:  HO_ID.
        LOCALITY: LOCAL.
        TYPE: INTEGER.
        USE: BOTH.
        MAKES:
             MESSAGE:  HANDOVER
             MESSAGE:  STATE_UPDATE
             MESSAGE:  TERMINATION
             MESSAGE:  TRACK_INITIATION
             MESSAGE:  TRACK_TERMINATION.
        INPUT TO:
             ALPHA:  TRACK_INITIATE.
        OUTPUT FROM:
             ALPHA:  GHOST_TERMINATION
             ALPHA:  LOW_TERMINATION
             ALPHA:  REDUN_TERMINATION
             ALPHA:  UPDATE_STATE.
        RECORDED BY:
             VALIDATION_POINT:  C2_IMAGE_HANDOVER.
        REFERRED BY:
             R_NET:  CC_RESPONSE.

DATA:  IMAGE_ID.
        LOCALITY: GLOBAL.
        TYPE: INTEGER.
        USE: BOTH.
        ASSOCIATED WITH:
             ENTITY_CLASS:  IMAGE.
        INPUT TO:
             ALPHA:  ASSIGN_RATE
             ALPHA:  CREATE_STATE_FILE
             ALPHA:  GHOST_TERMINATION
             ALPHA:  LOW_TERMINATION
             ALPHA:  PICK_CANDIDATES
             ALPHA:  REDUN_TERMINATION
             ALPHA:  UPDATE_STATE.
        OUTPUT FROM:
             ALPHA:  TRACK_INITIATE.
        REFERRED BY:
             R_NET:  CC_RESPONSE
             R_NET:  RESPONSE_TO_RADAR
             SUBNET:  EXTRACT_MEASUREMENT
             SUBNET:  FORM_FRAME.

DATA:  INITIAL_COVARIANCE.
```

```
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: BOTH.
        MAKES:
             MESSAGE:  HANDOVER.
        INPUT TO:
             ALPHA:  TRACK_INITIATE.

DATA:  INITIAL_STATE.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: BOTH.
        MAKES:
             MESSAGE:  HANDOVER
             MESSAGE:  TRACK_INITIATION.
        INPUT TO:
             ALPHA:  TRACK_INITIATE.

DATA:  IST.
        DESCRIPTION: "INITIAL START TIME FOR THE FRAME.".
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: BOTH.
        INPUT TO:
             ALPHA:  MAKE_COMMAND.
        OUTPUT FROM:
             ALPHA:  INITIALIZE_SKED_R
             ALPHA:  MAKE_COMMAND.

DATA:  LAST_ALLOCATE.
        LOCALITY: GLOBAL.
        TYPE: REAL.
        USE: BOTH.
        INPUT TO:
             ALPHA:  MAKE_ALLOCATION.
        OUTPUT FROM:
             ALPHA:  MAKE_ALLOCATION.

DATA:  LAST_PULSE.
        LOCALITY: GLOBAL.
        TYPE: REAL.
        USE: BOTH.
        ASSOCIATED WITH:
             ENTITY_TYPE:  IMAGE_IN_TRACK.
        INPUT TO:
             ALPHA:  SET_LAST.
        OUTPUT FROM:
             ALPHA:  SET_LAST.
        REFERRED BY:
             R_NET:  SKED_R.

DATA:  LENGTH_OF_RECEIVE.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
             DATA:  RECEIVE_INFORMATION.
```

```
DATA:  LOW_ELEVATION.
     LOCALITY: LOCAL.
     TYPE: BOOLEAN.
     USE: BOTH.
     OUTPUT FROM:
          ALPHA:  LOW_ELEVATION_DETERMINATION.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  RESPONSE_TO_RADAR.

DATA:  MODE.
     LOCALITY: GLOBAL.
     RANGE: "ENGAGED,STANDBY".
     TYPE: ENUMERATION.
     USE: BOTH.
     OUTPUT FROM:
          ALPHA:  ENGAGEMENT_INITIATION
          ALPHA:  TERM_ENGAGEMENT.
     REFERRED BY:
          R_NET:  RADAR_SUMMARY
          R_NET:  SKED_R.

DATA:  NOISE_LEVEL.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: GAMMA.
     INCLUDED IN:
          DATA:  T1_T2_RECORD
          DATA:  T3_RECORD.

DATA:  PRIORITY.
     LOCALITY: GLOBAL.
     TYPE: REAL.
     USE: BOTH.
     ORDERS:
          FILE:  CANDIDATE.
     CONTAINED IN:
          FILE:  CANDIDATE.
     OUTPUT FROM:
          ALPHA:  PICK_CANDIDATES.
     TRACED FROM:
          DECISION:  TRACK_PERFORMANCE_ALLOCATION
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS.

DATA:  PULSE_ID.
     LOCALITY: GLOBAL.
     TYPE: INTEGER.
     USE: BOTH.
     ASSOCIATED WITH:
          ENTITY_CLASS:  PULSE.
     OUTPUT FROM:
          ALPHA:  PICK_COMMAND.
     REFERRED BY:
          R_NET:  RESPONSE_TO_RADAR.
```

```
DATA:  PULSE_TYPE.
     LOCALITY: GLOBAL.
     RANGE: "T1,T2,T3".
     TYPE: ENUMERATION.
     USE: BOTH.
     ASSOCIATED WITH:
          ENTITY_CLASS:  PULSE.
     INPUT TO:
          ALPHA:  SUM_ENERGY.
     OUTPUT FROM:
          ALPHA:  PICK_COMMAND.
     REFERRED BY:
          R_NET:  RESPONSE_TO_RADAR.

DATA:  RADAR_CLOCK.
     LOCALITY: GLOBAL.
     TYPE: REAL.
     USE: BOTH.
     INPUT TO:
          ALPHA:  COMPLETE_SUMMARY.
     OUTPUT FROM:
          ALPHA:  UPDATE_RADAR_CLOCK.

DATA:  RADAR_CLOCK_TIME.
     LOCALITY: LOCAL.
     RESOLUTION: 6.25E-9.
     TYPE: REAL.
     UNITS: SECONDS.
     USE: BOTH.
     MAKES:
          MESSAGE:  R_CLOCK_MESSAGE.
     INPUT TO:
          ALPHA:  UPDATE_RADAR_CLOCK.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_RADAR_DPS_IFS_PARAGRAPH_3_2_9_FUNCTIONAL_REQUIREMENTS.

DATA:  RADAR_MEASUREMENT.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: BETA.
     INPUT TO:
          ALPHA:  GHOST_DETERMINATION
          ALPHA:  UPDATE_STATE.
     OUTPUT FROM:
          ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
          ALPHA:  T3_MEASUREMENT_EXTRACTION.

DATA:  RADAR_TYPE.
     LOCALITY: LOCAL.
     RANGE: "T1,T2,T3".
     TYPE: ENUMERATION.
     USE: BOTH.
     MAKES:
          MESSAGE:  T1_T2_COMMAND
          MESSAGE:  T1_T2_RETURN
          MESSAGE:  T3_COMMAND
          MESSAGE:  T3_RETURN.
```

```
      OUTPUT FROM:
            ALPHA:  PICK_COMMAND.
      RECORDED BY:
            VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
      REFERRED BY:
            R_JET:  RESPONSE_TO_RADAR
            R_JET:  XMIT_R.

DATA:  RANGE_MARK_GENERATION_TECHNIQUE.
      LOCALITY: LOCAL.
      TYPE: INTEGER.
      USE: GAMMA.
      INCLUDED IN:
            DATA:  T1_T2_GATE_DATA.

DATA:  RANGE_MARK_INFORMATION.
      USE: BETA.
      INCLUDES:
            DATA:  RANGE_MARK_TIME
            DATA:  SIGNAL_AMPLITUDE.
      INCLUDED IN:
            DATA:  T1_T2_RECORD
            DATA:  T3_RECORD.

DATA:  RANGE_MARK_TECHNIQUE.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: GAMMA.
      INCLUDED IN:
            DATA:  T3_GATE_DATA.

DATA:  RANGE_MARK_TIME.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: GAMMA.
      INCLUDED IN:
            DATA:  RANGE_MARK_INFORMATION.

DATA:  REASON_FOR_DROP.
      LOCALITY: LOCAL.
      RANGE: "GHOST,REDUNDANT,LOW,CC_COMMANDED_TO_DROP".
      TYPE: ENUMERATION.
      USE: BOTH.
      MAKES:
            MESSAGE:  TRACK_TERMINATION.
      OUTPUT FROM:
            ALPHA:  GHOST_TERMINATION
            ALPHA:  LOW_TERMINATION
            ALPHA:  REDUN_TERMINATION
            ALPHA:  TERM_TRACK.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
```

```
          TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS.

DATA:  REASON_FOR_TRANSMISSION_FAILURE.
       LOCALITY: LOCAL.
       RANGE:
        "PRE_EMPTED_TRANSMISSION,
         RECEIVE_WINDOW_OVERLAP,
         TRANSMIT_WINDOW_OVERLAP,
         INSUFFICIENT_TRANSMISSION_TIME,
         RADAR_COMMAND_INCONSISTENCY,
         TRANSMIT_START_TIME_EXCEEDED".
       TYPE: ENUMERATION.
       USE: BOTH.
       INCLUDED IN:
            DATA:  T1T2RTN_ERROR_REPORT
            DATA:  T3RTN_ERROR_REPORT.

DATA:  RECEIVE_INFORMATION.
       USE: BETA.
       INCLUDES:
            DATA:  LENGTH_OF_RECEIVE
            DATA:  RECEIVE_START_TIME
            DATA:  RECEIVER_GAIN_SETTING.
       INCLUDED IN:
            DATA:  T1_T2_TRANSMIT
            DATA:  T3_TRANSMIT.

DATA:  RECEIVE_START_TIME.
       LOCALITY: LOCAL.
       TYPE: REAL.
       USE: GAMMA.
       INCLUDED IN:
            DATA:  RECEIVE_INFORMATION.

DATA:  RECEIVE_STOP.
       LOCALITY: GLOBAL.
       TYPE: REAL.
       USE: BOTH.
       ASSOCIATED WITH:
            ENTITY_TYPE:  T1_T2_PULSE
            ENTITY_TYPE:  T3_PULSE.
       INPUT TO:
            ALPHA:  REMEMBER_STOP.
       OUTPUT FROM:
            ALPHA:  FORM_T1_T2
            ALPHA:  FORM_T3.
       REFERRED BY:
            SUBNET:  MISSING_RETURNS.

DATA:  RECEIVER_GAIN_SETTING.
       LOCALITY: LOCAL.
       TYPE: REAL.
       USE: GAMMA.
       INCLUDED IN:
            DATA:  RECEIVE_INFORMATION.

DATA:  RECORD_FOUND
            (* A PREDEFINED DATA ITEM WHICH IS SET TO EITHER
```

```
                    TRUE OR FALSE AFTER EACH SELECT ON A FILE IN
                    A BETA OR GAMMA.  RECORD_FOUND IS SET TO TRUE
                    IF A FILE RECORD SATISFYING THE SELECTION
                    CRITERION IS LOCATED.  OTHERWISE, RECORD_FOUND
                    IS ASSIGNED THE VALUE FALSE. *),
        INITIAL_VALUE: FALSE,
        LOCALITY: LOCAL,
        TYPE: BOOLEAN,
        USE: BOTH,
        OUTPUT FROM:
             ALPHA:  SELECT_COMMAND,
        REFERRED BY:
             R_NET:  XMIT_R.

DATA:  REDUNDANT_IMAGE,
        LOCALITY: LOCAL,
        TYPE: BOOLEAN,
        USE: BOTH,
        OUTPUT FROM:
             ALPHA:  REDUN_DETERMINATION,
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS,
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

DATA:  RESOURCES,
        LOCALITY: LOCAL,
        TYPE: REAL,
        USE: BETA,
        MAKES:
             MESSAGE:  RADAR_USAGE,
        OUTPUT FROM:
             ALPHA:  SUM_USAGE.

DATA:  RR_ORDER_ID,
        LOCALITY: LOCAL,
        TYPE: INTEGER,
        USE: BOTH,
        MAKES:
             MESSAGE:  T1_T2_COMMAND
             MESSAGE:  T1_T2_RETURN
             MESSAGE:  T3_COMMAND
             MESSAGE:  T3_RETURN,
        OUTPUT FROM:
             ALPHA:  PICK_COMMAND,
        REFERRED BY:
             R_NET:  RESPONSE_TO_RADAR.

DATA:  RR_ORDER_IDC,
        INITIAL_VALUE: 0,
        LOCALITY: GLOBAL,
        TYPE: INTEGER,
        USE: BOTH,
        INPUT TO:
             ALPHA:  PICK_COMMAND,
```

```
        OUTPUT FROM:
             ALPHA:  PICK_COMMAND.

DATA:  SIGNAL_AMPLITUDE.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
             DATA:  RANGE_MARK_INFORMATION.

DATA:  SIGNAL_PROCESSING_MODE.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: GAMMA.
        INCLUDED IN:
             DATA:  T1_T2_GATE_DATA
             DATA:  T3_GATE_DATA.

DATA:  START_TIME.
        LOCALITY: GLOBAL.
        TYPE: REAL.
        USE: BOTH.
        ORDERS:
             FILE:  COMMAND.
        CONTAINED IN:
             FILE:  COMMAND.
        INPUT TO:
             ALPHA:  FIND_CONFLICT
             ALPHA:  PICK_COMMAND
             ALPHA:  SET_LAST.
        OUTPUT FROM:
             ALPHA:  MAKE_COMMAND.

DATA:  STATE.
        LOCALITY: GLOBAL.
        TYPE: REAL.
        USE: BETA.
        ASSOCIATED WITH:
             ENTITY_TYPE:  IMAGE_IN_TRACK.
        INPUT TO:
             ALPHA:  CREATE_STATE_FILE
             ALPHA:  REDUN_DETERMINATION.
        OUTPUT FROM:
             ALPHA:  TRACK_INITIATE
             ALPHA:  UPDATE_STATE.
        TRACED FROM:
             ORIGINATING_REQUIREMENT:
             TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS.

DATA:  STATE_DATA.
        LOCALITY: LOCAL.
        TYPE: REAL.
        USE: BETA.
        CONTAINED IN:
             FILE:  STATE_FILE.
        OUTPUT FROM:
             ALPHA:  MAKE_ALLOCATION.
```

```
DATA:  STATE_ID.
     LOCALITY: LOCAL.
     TYPE: INTEGER.
     USE: BETA.
     CONTAINED IN:
          FILE:  STATE_FILE.

DATA:  STOP_TIME.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: BOTH.
     OUTPUT FROM:
          ALPHA:  REMEMBER_STOP.
     REFERRED BY:
          SUBNET:  MISSING_RETURNS.

DATA:  SUMMARY_RATE.
     INITIAL_VALUE: 0.3.
     LOCALITY: GLOBAL.
     TYPE: REAL.
     USE: BOTH.
     DELAYS:
          EVENT:  SUMMARIZE.

DATA:  TARGET_ID.
     LOCALITY: GLOBAL.
     TYPE: INTEGER.
     USE: BOTH.
     ASSOCIATED WITH:
          ENTITY_CLASS:  PULSE.
     OUTPUT FROM:
          ALPHA:  PICK_COMMAND.
     RECORDED BY:
          VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
     REFERRED BY:
          R_NET:  RESPONSE_TO_RADAR
          SUBNET:  EXTRACT_MEASUREMENT.

DATA:  TEMP.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: BOTH.
     INPUT TO:
          ALPHA:  FIND_CONFLICT.
     OUTPUT FROM:
          ALPHA:  INITIALIZE_SKED_R.
     REFERRED BY:
          R_NET:  SKED_R.

DATA:  THRESHOLD_DOWN_CROSSING_TIME.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: GAMMA.
     INCLUDED IN:
          DATA:  WAKE_ARRAY.

DATA:  THRESHOLD_TYPE.
     LOCALITY: LOCAL.
```

```
      TYPE: REAL.
      USE: GAMMA.
      INCLUDED IN:
            DATA:  T1_T2_GATE_DATA
            DATA:  T3_GATE_DATA.

DATA:  THRESHOLD_UP_CROSSING_TIME.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: GAMMA.
      INCLUDED IN:
            DATA:  WAKE_ARRAY.

DATA:  TIME_OF_DROP.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BOTH.
      MAKES:
            MESSAGE:  TRACK_TERMINATION.
      OUTPUT FROM:
            ALPHA:  GHOST_TERMINATION
            ALPHA:  LOW_TERMINATION
            ALPHA:  REDUN_TERMINATION
            ALPHA:  TERM_TRACK.

DATA:  TIME_OF_INITIATION.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BOTH.
      MAKES:
            MESSAGE:  TRACK_INITIATION.
      OUTPUT FROM:
            ALPHA:  TRACK_INITIATE.

DATA:  TRACK_RATE.
      LOCALITY: GLOBAL.
      TYPE: REAL.
      USE: BOTH.
      ASSOCIATED WITH:
            ENTITY_TYPE:  IMAGE_IN_TRACK.
      OUTPUT FROM:
            ALPHA:  ASSIGN_RATE
            ALPHA:  TRACK_INITIATE.
      REFERRED BY:
            R_NET:  SKED_R.

DATA:  TRANSMIT_START.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BOTH.
      MAKES:
            MESSAGE:  T1_T2_COMMAND
            MESSAGE:  T3_COMMAND.
      INPUT TO:
            ALPHA:  FORM_T1_T2
            ALPHA:  FORM_T3.
      OUTPUT FROM:
            ALPHA:  PICK_COMMAND.
```

```
        RECORDED BY:
             VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.

DATA:  T1_T2_GATE_DATA.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: BETA.
     INCLUDES:
          DATA:  ACCEPTANCE_THRESHOLD
          DATA:  GATE_LENGTH
          DATA:  GATE_START_TIME
          DATA:  RANGE_MARK_GENERATION_TECHNIQUE
          DATA:  SIGNAL_PROCESSING_MODE
          DATA:  THRESHOLD_TYPE.
     CONTAINED IN:
          FILE:  T1_T2_GATE.

DATA:  T1_T2_RECEIVE.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: BOTH.
     INCLUDES:
          DATA:  ALPHA_ERROR
          DATA:  BETA_ERROR
          DATA:  T1T2RTN_ERROR_REPORT
          DATA:  MAKE_ARRAY.
     MAKES:
          MESSAGE:  T1_T2_RETURN.
     INPUT TO:
          ALPHA:  T1_T2_MEASUREMENT_EXTRACTION.

DATA:  T1_T2_RECORD.
     LOCALITY: LOCAL.
     TYPE: REAL.
     USE: BETA.
     INCLUDES:
          DATA:  NOISE_LEVEL
          DATA:  RANGE_MARK_INFORMATION.
     CONTAINED IN:
          FILE:  T1_T2_DATA.

DATA:  T1_T2_TRANSMIT.
     TYPE: REAL.
     USE: BETA.
     INCLUDES:
          DATA:  ALPHA_PHASE_TAPER
          DATA:  BETA_PHASE_TAPER
          DATA:  RECEIVE_INFORMATION.
     MAKES:
          MESSAGE:  T1_T2_COMMAND.
     OUTPUT FROM:
          ALPHA:  FORM_T1_T2.

DATA:  T1_T2_WINDOW_DATA.
     LOCALITY: GLOBAL.
     TYPE: REAL.
     USE: BOTH.
     CONTAINED IN:
```

```
            FILE:  T1_T2_WINDOW.
      INPUT TO:
            ALPHA:  SET_LOST.

DATA:  T1_T2_XMIT.
      LOCALITY: GLOBAL.
      TYPE: REAL.
      USE: BOTH.
      ASSOCIATED WITH:
            ENTITY_TYPE:  T1_T2_PULSE.
      INPUT TO:
            ALPHA:  SET_LOST.
      OUTPUT FROM:
            ALPHA:  FORM_T1_T2.

DATA:  T1T2RTN_ERROR_REPORT.
      USE: BOTH.
      INCLUDES:
            DATA:  REASON_FOR_TRANSMISSION_FAILURE.
      INCLUDED IN:
            DATA:  T1_T2_RECEIVE.

DATA:  T3_GATE_DATA.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BETA.
      INCLUDES:
            DATA:  ACCEPTANCE_THRESHOLD
            DATA:  GATE_LENGTH
            DATA:  GATE_START_TIME
            DATA:  RANGE_MARK_TECHNIQUE
            DATA:  SIGNAL_PROCESSING_MODE
            DATA:  THRESHOLD_TYPE.
      CONTAINED IN:
            FILE:  T3_GATE.

DATA:  T3_RECEIVE.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BOTH.
      INCLUDES:
            DATA:  ALPHA_ERROR
            DATA:  BETA_ERROR
            DATA:  T3RTN_ERROR_REPORT
            DATA:  WAKE_ARRAY.
      MAKES:
            MESSAGE:  T3_RETURN.
      INPUT TO:
            ALPHA:  T3_MEASUREMENT_EXTRACTION.

DATA:  T3_RECORD.
      LOCALITY: LOCAL.
      TYPE: REAL.
      USE: BETA.
      INCLUDES:
            DATA:  NOISE_LEVEL
            DATA:  RANGE_MARK_INFORMATION.
      CONTAINED IN:
```

```
          FILE:  T3_DATA.

DATA:  T3_TRANSMIT.
     TYPE: REAL.
     USE: BETA.
     INCLUDES:
          DATA:  ALPHA_PHASE_TAPER
          DATA:  BETA_PHASE_TAPER
          DATA:  RECEIVE_INFORMATION.
     MAKES:
          MESSAGE:  T3_COMMAND.
     OUTPUT FROM:
          ALPHA:  FORM_T3.

DATA:  T3_WINDOW_DATA.
     LOCALITY: GLOBAL.
     TYPE: REAL.
     USE: BETA.
     CONTAINED IN:
          FILE:  T3_WINDOW.
     INPUT TO:
          ALPHA:  SET_LOST.

DATA:  T3_XMIT.
     LOCALITY: GLOBAL.
     TYPE: REAL.
     USE: BOTH.
     ASSOCIATED WITH:
          ENTITY_TYPE:  T3_PULSE.
     INPUT TO:
          ALPHA:  SET_LOST.
     OUTPUT FROM:
          ALPHA:  FORM_T3.

DATA:  T3RTN_ERROR_REPORT.
     USE: BOTH.
     INCLUDES:
          DATA:  REASON_FOR_TRANSMISSION_FAILURE.
     INCLUDED IN:
          DATA:  T3_RECEIVE.

DATA:  VALID_RETURN.
     LOCALITY: LOCAL.
     TYPE: BOOLEAN.
     USE: BOTH.
     OUTPUT FROM:
          ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
          ALPHA:  T3_MEASUREMENT_EXTRACTION.
     REFERRED BY:
          R_NET:  RESPONSE_TO_RADAR.

DATA:  WAKE_ARRAY.
     USE: BOTH.
     INCLUDES:
          DATA:  AVERAGE_SIGNAL_POWER
          DATA:  THRESHOLD_DOWN_CROSSING_TIME
          DATA:  THRESHOLD_UP_CROSSING_TIME.
     INCLUDED IN:
```

```
            DATA:  T1_T2_RECEIVE
            DATA:  T3_RECEIVE.

DATA:  WAVEFORM,
     LOCALITY: GLOBAL,
     RANGE: "T1,T2,T3",
     TYPE: ENUMERATION,
     USE: BOTH,
     ASSOCIATED WITH:
          ENTITY_TYPE:  IMAGE_IN_TRACK,
     INPUT TO:
          ALPHA:  PICK_CANDIDATES
          (*INSTANCES OF ENTITY_TYPE IMAGE_IN_TRACK*)
          ALPHA:  UPDATE_STATE,
     OUTPUT FROM:
          ALPHA:  TRACK_INITIATE
          ALPHA:  UPDATE_STATE.

DATA:  WF_CHARACTERISTICS,
     LOCALITY: GLOBAL,
     TYPE: REAL,
     USE: BOTH,
     CONTAINED IN:
          FILE:  WAVEFORM_TABLE,
     RECORDED BY:
          VALIDATION_POINT:  STARTING_POINT,

DATA:  WF_NAME,
     LOCALITY: GLOBAL,
     RANGE: "T1,T2,T3",
     TYPE: ENUMERATION,
     USE: BOTH,
     CONTAINED IN:
          FILE:  WAVEFORM_TABLE,
     RECORDED BY:
          VALIDATION_POINT:  STARTING_POINT,

DATA:  XMIT_START,
     LOCALITY: GLOBAL,
     TYPE: REAL,
     USE: BOTH,
     ASSOCIATED WITH:
          ENTITY_CLASS:  PULSE,
     INPUT TO:
          ALPHA:  PICK_COMMAND,
     OUTPUT FROM:
          ALPHA:  PICK_COMMAND,

DECISION:  RADAR_RESOURCE_CONTROL_R1,
     ALTERNATIVES:
  "1. THE ALLOCATOR SHALL ASSIGN TRACK RATES SUCH THAT THE
     CUMULATIVE SUM OF THE ENERGY FOR EACH IMAGE OVER THE
     ENGAGEMENT (THE PRODUCT OF ALLOCATED PULSE RATE,
     ENERGY PER PULSE, AND DURATION OF ALLOCATION) SHALL
     NOT EXCEED (TBD) JOULES.
   2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED
     ACROSS THE INTERFACE TO  THE RADAR SHALL NOT EXCEED
     (TBD) JOULES.
```

```
    3. THE ENERGY REQUIRED BY THE RADAR COMMANDS ACTED UPON
       BY THE RADAR AS DETECTED BY THE DPS IN THE RETURN
       MESSAGES, SHALL NOT EXCEED (TBD) JOULES.".
       CHOICE:
    "2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED
       ACROSS THE INTERFACE TO THE RADAR SHALL NOT EXCEED
       (TBD) JOULES SHALL BE TESTED FOR COMPLIANCE AT THE
       INPUT TO THE OUTPUT INTERFACE: RADAR_OUT.".
       PROBLEM:
    "DPSPR PARAGRAPH 3.2.4(B), STATEMENT ('THE DPS SHALL
    ALLOCATE RADAR COMMANDS SO THAT NOT MORE THAN (TBD) JOULES
    ARE COMMANDED PER IMAGE,...') ALLOWS FOR THREE POSSIBLE
    INTERPRETATIONS IN DETERMINING THE POINT AT WHICH THE
    PERFORMANCE_REQUIREMENT TEST IS APPLIED.".
       TRACES TO:
            PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE
            VALIDATION_PATH:  RADAR_COMMAND_OUTPUT
            VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
       TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.

DECISION:  RADAR_RESOURCE_CONTROL_R2.
       ALTERNATIVES:
    "1. THE RADAR POWER CONSTRAINTS COULD BE APPLIED OVER THE TIME
    INTERVAL OF THE ENGAGEMENT.
     2. THE RADAR POWER CONSTRAINTS COULD BE APPLIED OVER THE TIME
    INTERVAL OF THE RADAR FRAME.
     3. THE RADAR POWER CONSTRAINT COULD BE APPLIED OVER A FINITE
    TIME INTERVAL SPECIFIED IN SECONDS.".
       CHOICE:
    "3. THE RADAR POWER CONSTRAINT SHALL APPLY OVER A TIME INTERVAL
    OF ONE SECOND AND SHALL BE TESTED FOR COMPLIANCE DURING THE
    ENGAGEMENT FOR ALL IMAGES IN TRACK WITHIN EACH ONE SECOND INTERVAL
    FOR WHICH RADAR COMMANDS ARE ISSUED.".
       PROBLEM:
    "DPSPR PARAGRAPH 3.2.4(3), STATEMENT ('THE DPS SHALL ALLOCATE
    RADAR COMMANDS SO THAT NOT MORE THAN___, NOR MORE THAN (TBD) KILO-
    WATTS___FOR ALL IMAGES IN TRACK.') DOES NOT SPECIFY THE TIME
    INTERVAL OVER WHICH THE RADAR ENERGY MUST BE ACCUMULATED TO TEST
    FOR COMPLIANCE WITH THE RADAR POWER CONSTRAINT.".
       TRACES TO:
            PERFORMANCE_REQUIREMENT:  RADIATED_POWER
            VALIDATION_PATH:  RADAR_COMMAND_OUTPUT
            VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
       TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.

DECISION:  RADAR_SCHEDULER_PRIORITIZATION.
       ALTERNATIVES:
            "1. SCHEDULE PULSE_BY_PULSE.  THIS WOULD SIMPLIFY THE
              NETS BUT WOULD ORVIATE OPTIMIZATION.
            2. OPTIMIZE OVER THE ENTIRE FRAME.  TAKING THE FRAME
              AS A WHOLE GIVES BEST RESULTS BUT REQUIRES WEIGHTING
              FACTORS FOR PULSE ENSEMBLES.
            3. PRIORITIZE PULSES SUCH THAT ANY PULSE OF HIGH
              PRIORITY BEATS ALL PULSES OF LOWER.  THIS IS
```

```
                SUBOPTIMAL, BUT REALIZABLE BOTH IN THE SPEC AND IN
                THE SOFTWARE DESIGN.  NO A PRIORI WEIGHTS NEEDED.".
    CHOICE: "OPTION 3, PRIORITIZED PULSES".
    PROBLEM:
        "OPTIMIZATION OF RADAR USAGE REQUIRES A FINITE RADAR FRAME.
        THIS IMPLIES A PRIORITIZATION SCHEME FOR INTENDED ORDERS.".
    TRACES TO:
        R_NET:  SKED_R
        R_NET:  XMIT_R.
    TRACED FROM:
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.

DECISION:  SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK.
    ALTERNATIVES:
            "1. SYNCHRONOUS TRACKING (OR RESPONSIVE)
                REQUIRES THE LAST RADAR RETURN ON
                AN IMAGE BE USED TO
                PRODUCE THE NEXT RADAR
                ORDER.
             2. ASYNCHRONOUS TRACKING (OR AUTOGENIC<
                ALLOWS A TRACK PULSE
                TO BE SENT USING WHAT EVER
                STATE IS IN THE DATA BASE.".
    CHOICE:
      "ASYNCHRONOUS TRACKING IS
        SELECTED TO MAXIMIZE THE
        ALLOWED OR TIME RESPONSE
        FOR PROCESSING RADAR RETURNS.
        THIS DOES NOT PROHIBIT A RESPONSIVE
        TRACKING IMPLEMENTATION.".
    PROBLEM:
       "TRACKING CAN BE EXPRESSED AS
            SYNCHRONOUS OR ASYNCHRONOUS.".
    TRACES TO:
        ALPHA:  PICK_CANDIDATES.
    TRACED FROM:
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS.

DECISION:  TRACK_PERFORMANCE_ALLOCATION.
    CHOICE:
        "1. ALLOCATION WILL BE PERFORMED TO CONSTRAIN
            IMAGE STATES AND RATES AT ALL TIMES.
        2. PULSE SCHEDULING WILL BE CONSTRAINED
            BY A REALTIONSHIP BETWEEN IMAGE
            STATES, ITS TRACK RATE,
            AND THE RADAR CONSTRAINTS.
        3. DEGHOSTING WILL BE A FUNCTION OF
            RADAR MEASUREMENTS ONLY.
        4. >UPDATE STATE> WILL BE CONSTRAINED
            BY ITS DIFFERENCE
            IN BETA AND CEP FROM A
            >PERFECT FILTER>.
        5. REDUNDANT IMAGE ELIMINATION
            PERFORMANCE
            WILL BE EXPRESSED IN TERMS OF
            STATES ONLY.".
```

```
        PROBLEM:
            "TRACK ACCURACY IS A JOINT FUNCTION OF
             THE TRACK RATE, SUCCESSFUL SCHEDULING,
             ACCURATE PULSE COMMANDS, AND ACCURATE
             PROCESSING OF THE RADAR RETURN".
        TRACES TO:
              ALPHA:  FIND_CONFLICT
              ALPHA:  GHOST_DETERMINATION
              ALPHA:  REBUN_DETERMINATION
              ALPHA:  UPDATE_STATE
              DATA:  PRIORITY.
        TRACED FROM:
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_2_A_PERFORMANCE_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_A_PERFORMANCE_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_B_PERFORMANCE_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_C_PERFORMANCE_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_D_PERFORMANCE_REQUIREMENTS.

ENTITY_CLASS:  IMAGE.
        ASSOCIATES:
              DATA:  ENTRY_TIME
              DATA:  IMAGE_ID.
        COMPOSED OF:
              ENTITY_TYPE:  DROPPED_IMAGE
              ENTITY_TYPE:  IMAGE_IN_TRACK.
        CREATED BY:
              ALPHA:  TRACK_INITIATE.
        TRACED FROM:
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
```

```
            R_NET:  CC_RESPONSE
            R_NET:  RESPONSE_TO_RADAR
            SUBNET:  EXTRACT_MEASUREMENT
            SUBNET:  RECORD_DROP.

ENTITY_CLASS:  PULSE.
      ASSOCIATES:
            DATA:  PULSE_ID
            DATA:  PULSE_TYPE
            DATA:  TARGET_ID
            DATA:  XMIT_START.
      COMPOSED OF:
            ENTITY_TYPE:  LAST_PULSE
            ENTITY_TYPE:  RETURNED_PULSE
            ENTITY_TYPE:  T1_T2_PULSE
            ENTITY_TYPE:  T3_PULSE.
      CREATED BY:
            ALPHA:  PICK_COMMAND.
      DESTROYED BY:
            ALPHA:  DROP_LAST
            ALPHA:  DROP_PULSE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  RESPONSE_TO_RADAR
            SUBNET:  EXTRACT_MEASUREMENT
            SUBNET:  MISSING_RETURNS.

ENTITY_TYPE:  DROPPED_IMAGE.
      ASSOCIATES:
            FILE:  TERMINATOR.
      COMPOSES:
            ENTITY_CLASS:  IMAGE.
      SET BY:
            ALPHA:  SET_DROP.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
```

```
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_F_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            SUBNET:  RECORD_DROP.

ENTITY_TYPE:  IMAGE_IN_TRACK.
      ASSOCIATES:
            DATA:  COVARIANCE
            DATA:  LAST_PULSE
            DATA:  STATE
            DATA:  TRACK_RATE
            DATA:  WAVEFORM.
      COMPOSES:
            ENTITY_CLASS:  IMAGE.
      SET BY:
            ALPHA:  TRACK_INITIATE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  CONTROL_RESOURCES
            R_NET:  RESPONSE_TO_RADAR
            R_NET:  SKED_R
            SUBNET:  FORM_FRAME
            SUBNET:  RECORD_DROP.

ENTITY_TYPE:  LAST_PULSE.
      ASSOCIATES:
            DATA:  ACCOUNTED_FOR.
      COMPOSES:
            ENTITY_CLASS:  PULSE.
      SET BY:
            ALPHA:  SET_LAST.
      REFERRED BY:
            R_NET:  CONTROL_RESOURCES
            SUBNET:  EXTRACT_MEASUREMENT
            SUBNET:  MISSING_RETURNS.

ENTITY_TYPE:  RETURNED_PULSE.
      ASSOCIATES:
            DATA:  ACCOUNTED_FOR.
      COMPOSES:
            ENTITY_CLASS:  PULSE.
```

```
      SET BY:
            ALPHA:  SET_PULSE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  CONTROL_RESOURCES
            R_NET:  RADAR_SUMMARY
            SUBNET:  EXTRACT_MEASUREMENT
            SUBNET:  MISSING_RETURNS.

ENTITY_TYPE:  T1_T2_PULSE.
      ASSOCIATES:
            DATA:  RECEIVE_STRB
            DATA:  T1_T2_XMIT
            FILE:  T1_T2_WINDOW.
      COMPOSES:
            ENTITY_CLASS:  PULSE.
      SET BY:
            ALPHA:  FORM_T1_T2.
      REFERRED BY:
            SUBNET:  EXTRACT_MEASUREMENT
            SUBNET:  MISSING_RETURNS.

ENTITY_TYPE:  T3_PULSE.
      ASSOCIATES:
            DATA:  RECEIVE_STRB
            DATA:  T3_XMIT
            FILE:  T3_WINDOW.
      COMPOSES:
            ENTITY_CLASS:  PULSE.
      SET BY:
            ALPHA:  FORM_T3.
      REFERRED BY:
            SUBNET:  EXTRACT_MEASUREMENT
            SUBNET:  MISSING_RETURNS.

EVENT:  ALLOCATE.
      ENABLES:
            R_NET:  CONTROL_RESOURCES.
      REFERRED BY:
            R_NET:  CC_RESPONSE
            SUBNET:  RECORD_DROP.

EVENT:  SCHEDULE.
      ENABLES:
            R_NET:  SKED_R.
      DELAYED BY:
            DATA:  FRAME_RATE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
```

```
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  CC_RESPONSE
            R_NET:  XMIT_R.

EVENT:  SUMMARIZE.
      ENABLES:
            R_NET:  RADAR_SUMMARY.
      DELAYED BY:
            DATA:  SUMMARY_RATE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  CC_RESPONSE
            R_NET:  RADAR_SUMMARY.

EVENT:  XPB.
      DESCRIPTION: "TURNS ON R_NET XMIT_R FOR THE CURRENT FRAME.".
      ENABLES:
            R_NET:  XMIT_R.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  SKED_R
            R_NET:  XMIT_R.

FILE:  CANDIDATE.
      CONTAINS:
            DATA:  CANDIDATE_ENERGY
            DATA:  CANDIDATE_IMAGE_ID
            DATA:  CANDIDATE_WAVEFORM
            DATA:  PRIORITY.
      ORDERED BY:
            DATA:  PRIORITY.
      REFERRED BY:
            R_NET:  SKED_R.

FILE:  COMMAND.
      CONTAINS:
            DATA:  COMMAND_ENERGY
            DATA:  COMMAND_IMAGE_ID
```

```
            DATA:   COMMAND_WAVEFORM
            DATA:   START_TIME.
      INPUT TO:
            ALPHA:  PICK_COMMAND
            ALPHA:  SELECT_COMMAND.
      ORDERED BY:
            DATA:   START_TIME.

FILE:  STATE_FILE.
      LOCALITY: LOCAL.
      CONTAINS:
            DATA:   STATE_DATA
            DATA:   STATE_ID.
      INPUT TO:
            ALPHA:  ASSIGN_RATE
            ALPHA:  MAKE_ALLOCATION.
      OUTPUT FROM:
            ALPHA:  CREATE_STATE_FILE.

FILE:  TERMINATOR.
      CONTAINS:
            DATA:   DROP_REASON
            DATA:   DROP_TIME.
      ASSOCIATED WITH:
            ENTITY_TYPE:  DROPPED_IMAGE.
      OUTPUT FROM:
            ALPHA:  GHOST_TERMINATION
            ALPHA:  LOW_TERMINATION
            ALPHA:  REDUN_TERMINATION
            ALPHA:  TERM_TRACK.

FILE:  T1_T2_DATA.
      LOCALITY: LOCAL.
      CONTAINS:
            DATA:   T1_T2_RECORD.
      MAKES:
            MESSAGE:  T1_T2_RETURN.
      INPUT TO:
            ALPHA:  T1_T2_MEASUREMENT_EXTRACTION.

FILE:  T1_T2_GATE.
      LOCALITY: LOCAL.
      CONTAINS:
            DATA:   T1_T2_GATE_DATA.
      MAKES:
            MESSAGE:  T1_T2_COMMAND.
      OUTPUT FROM:
            ALPHA:  FORM_T1_T2.

FILE:  T1_T2_WINDOW.
      CONTAINS:
            DATA:   T1_T2_WINDOW_DATA.
      ASSOCIATED WITH:
            ENTITY_TYPE:  T1_T2_PULSE.
      OUTPUT FROM:
            ALPHA:  FORM_T1_T2.

FILE:  T3_DATA.
```

```
        LOCALITY: LOCAL.
        CONTAINS:
            DATA:  T3_RECORD.
        MAKES:
            MESSAGE:  T3_RETURN.
        INPUT TO:
            ALPHA:  T3_MEASUREMENT_EXTRACTION.

    FILE:  T3_GATE.
        LOCALITY: LOCAL.
        CONTAINS:
            DATA:  T3_GATE_DATA.
        MAKES:
            MESSAGE:  T3_COMMAND.
        OUTPUT FROM:
            ALPHA:  FORM_T3.

    FILE:  T3_WINDOW.
        CONTAINS:
            DATA:  T3_WINDOW_DATA.
        ASSOCIATED WITH:
            ENTITY_TYPE:  T3_PULSE.
        OUTPUT FROM:
            ALPHA:  FORM_T3.

    FILE:  WAVEFORM_TABLE.
        CONTAINS:
            DATA:  WF_CHARACTERISTICS
            DATA:  WF_NAME.
        INPUT TO:
            ALPHA:  PICK_CANDIDATES
            ALPHA:  SUM_ENERGY.
        OUTPUT FROM:
            ALPHA:  STARTER.
        RECORDED BY:
            VALIDATION_POINT:  STARTING_POINT.

    INPUT_INTERFACE:  CC_IN.
        CONNECTS TO:
            SUBSYSTEM:  SSC2.
        ENABLES:
            R_NET:  CC_RESPONSE.
        PASSES:
            MESSAGE:  HANDOVER
            MESSAGE:  MODE_CHANGE
            MESSAGE:  TERMINATION.
        TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS.
        REFERRED BY:
            R_NET:  CC_RESPONSE.

    INPUT_INTERFACE:  RADAR_CLOCK_IN.
        CONNECTS TO:
            SUBSYSTEM:  SSRADAR.
        ENABLES:
```

```
          R_NET:  RADAR_TIMING.
     PASSES:
          MESSAGE:  R_CLOCK_MESSAGE.
     REFERRED BY:
          R_NET:  RADAR_TIMING.

INPUT_INTERFACE:  RADAR_IN.
     DESCRIPTION:
          "THE RADIN INTERFACE PROVIDES THE MECHANISM THROUGH
           WHICH THE DPS RECEIVES RADAR SUBSYSTEM RETURNS IN
           RESPONSE TO RADAR COMMANDS ISSUED BY THE DPS THROUGH
           THE RADOUT INTERFACE. RADAR SUBSYSTEM RETURN MESSAGES
           SHALL COMPLY WITH REQUIREMENTS SPECIFIED IN THE TLS
           RADAR_DPS INTERFACE SPECIFICATION, PARAGRAPH_3_2_6.".
     ENTERED_BY: "H.A.HELTON, MAY. 3,1976.".
     CONNECTS TO:
          SUBSYSTEM:  SSRADAR.
     ENABLES:
          R_NET:  RESPONSE_TO_RADAR.
     IMPLEMENTS:
          VERSION:  ORIGINAL_PUBLICATION_DATED_AUGUST_1975.
     PASSES:
          MESSAGE:  T1_T2_RETURN
          MESSAGE:  T3_RETURN.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  RESPONSE_TO_RADAR.

MESSAGE:  ACKNOWLEDGEMENT.
     FORMED BY:
          ALPHA:  ACKNOWLEDGE.
     MADE BY:
          DATA:  COMMAND_ID.
     PASSED THROUGH:
          OUTPUT_INTERFACE:  CC_OUT.

MESSAGE:  HANDOVER.
     MADE BY:
          DATA:  COMMAND_ID
          DATA:  MC_ID
          DATA:  INITIAL_COVARIANCE
          DATA:  INITIAL_STATE.
     PASSED THROUGH:
          INPUT_INTERFACE:  CC_IN.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_1_B_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  MODE_CHANGE.
     MADE BY:
          DATA:  COMMAND_ID.
     PASSED THROUGH:
```

```
            INPUT_INTERFACE:  CC_IN.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_B_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  R_CLOCK_MESSAGE.
      MADE BY:
            DATA:  RADAR_CLOCK_TIME.
      PASSED THROUGH:
            INPUT_INTERFACE:  RADAR_CLOCK_IN.

MESSAGE:  RADAR_USAGE.
      FORMED BY:
            ALPHA:  COMPLETE_SUMMARY.
      MADE BY:
            DATA:  DATA_RECORD_TYPE
            DATA:  ENGAGEMENT_TIME
            DATA:  RESOURCES.
      PASSED THROUGH:
            OUTPUT_INTERFACE:  DATA_RECORD.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  STATE_UPDATE.
      FORMED BY:
            ALPHA:  FORM_UPDATE.
      MADE BY:
            DATA:  CURRENT_STATE
            DATA:  DATA_RECORD_TYPE
            DATA:  HO_ID.
      PASSED THROUGH:
            OUTPUT_INTERFACE:  DATA_RECORD.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  TERMINATION.
      MADE BY:
            DATA:  COMMAND_ID
            DATA:  HO_ID.
      PASSED THROUGH:
            INPUT_INTERFACE:  CC_IN.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  TRACK_INITIATION.
      FORMED BY:
            ALPHA:  TRACK_INITIATE.
      MADE BY:
            DATA:  DATA_RECORD_TYPE
            DATA:  HO_ID
            DATA:  INITIAL_STATE
            DATA:  TIME_OF_INITIATION.
      PASSED THROUGH:
```

```
            OUTPUT_INTERFACE:  DATA_RECORD.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  TRACK_TERMINATION.
      FORMED BY:
            ALPHA:  GHOST_TERMINATION
            ALPHA:  LAW_TERMINATION
            ALPHA:  REDUN_TERMINATION
            ALPHA:  TERM_TRACK.
      MADE BY:
            DATA:  DATA_RECORD_TYPE
            DATA:  MR_ID
            DATA:  REASON_FOR_DROP
            DATA:  TIME_OF_DROP.
      PASSED THROUGH:
            OUTPUT_INTERFACE:  DATA_RECORD.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  T1_T2_COMMAND.
      EQUATED TO:
            SYNONYM:  T1T2CMD.
      FORMED BY:
            ALPHA:  FORM_T1_T2.
      MADE BY:
            DATA:  RADAR_TYPE
            DATA:  RR_ORDER_ID
            DATA:  TRANSMIT_START
            DATA:  T1_T2_TRANSMIT
            FILE:  T1_T2_GATE.
      PASSED THROUGH:
            OUTPUT_INTERFACE:  RADAR_OUT.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  T1_T2_RETURN.
      EQUATED TO:
            SYNONYM:  T1T2RTN.
      MADE BY:
            DATA:  RADAR_TYPE
            DATA:  RR_ORDER_ID
            DATA:  T1_T2_RECEIVE
            FILE:  T1_T2_DATA.
      PASSED THROUGH:
            INPUT_INTERFACE:  RADAR_IN.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  T3_COMMAND.
```

```
      EQUATED TO:
            SYNONYM:  T3CMD.
      FORMED BY:
            ALPHA:  FORM_T3.
      MADE BY:
            DATA:  RADAR_TYPE
            DATA:  RR_ORDER_ID
            DATA:  TRANSMIT_START
            DATA:  T3_TRANSMIT
            FILE:  T3_GATE.
      PASSED THROUGH:
            OUTPUT_INTERFACE:  RADAR_OUT.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

MESSAGE:  T3_RETURN.
      EQUATED TO:
            SYNONYM:  T3RTN.
      MADE BY:
            DATA:  RADAR_TYPE
            DATA:  RR_ORDER_ID
            DATA:  T3_RECEIVE
            FILE:  T3_DATA.
      PASSED THROUGH:
            INPUT_INTERFACE:  RADAR_IN.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "ACTIONS: ACCEPT C2 MESSAGE,INITIATE TRACK ON IMAGE,
                      SEND RADAR ORDER
             INFORMATION: C2 MESSAGE %INITIATE TRACK COMMAND<,
                          HANDOVER IMAGE, RADAR ORDER".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  ENGAGEMENT_INITIATION
            ALPHA:  STARTER
            ALPHA:  TRACK_INITIATE
            ALPHA:  VALIDATE_HEADER
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  IMAGE_IN_TRACK
            INPUT_INTERFACE:  CC_IN
            MESSAGE:  HANDOVER
            OUTPUT_INTERFACE:  CC_ALT
            OUTPUT_INTERFACE:  RADAR_OUT
            R_NET:  CC_RESPONSE.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
```

```
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_1_A_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_1_B_FUNCTIONAL_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            MESSAGE:  HANDOVER
            MESSAGE:  MODE_CHANGE
            MESSAGE:  TERMINATION.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_1_B_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "   ACTION: SEND RADAR ORDER
                INFORMATION: RADAR, REDUNDANT IMAGE.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  INITIALIZE_SKED_R
            ALPHA:  REDUN_DETERMINATION
            ALPHA:  REDUN_TERMINATION
            DATA:  DROP_REASON
            DATA:  REASON_FOR_DROP
            DATA:  REDUNDANT_IMAGE
            ENTITY_CLASS:  IMAGE
            ENTITY_CLASS:  PULSE
            ENTITY_TYPE:  DROPPED_IMAGE
            EVENT:  SCHEDULE
            EVENT:  XMIT
            R_NET:  RESPONSE_TO_RADAR
            R_NET:  SKED_R
            R_NET:  XMIT_R
            SUBNET:  FORM_FRAME.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
```

```
            TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_A_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            DECISION:  TRACK_PERFORMANCE_ALLOCATION.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "  ACTIONS: SEND RADAR ORDER
               INFORMATION: GHOST IMAGE, RADAR ORDER.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  GHOST_DETERMINATION
            ALPHA:  GHOST_TERMINATION
            ALPHA:  INITIALIZE_SKED_R
            DATA:  DROP_REASON
            DATA:  GHOST_IMAGE
            DATA:  REASON_FOR_DROP
            ENTITY_CLASS:  IMAGE
            ENTITY_CLASS:  PULSE
            ENTITY_TYPE:  DROPPED_IMAGE
            EVENT:  SCHEDULE
            EVENT:  XRH
            R_NET:  RESPONSE_TO_RADAR
            R_NET:  SKED_R
            R_NET:  XMIT_R
            SUBNET:  FORM_FRAME.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  FIND_CONFLICT
            ALPHA:  GHOST_DETERMINATION
            ALPHA:  REDUN_DETERMINATION
            ALPHA:  UPDATE_STATE
            DATA:  PRIORITY
            DECISION:  TRACK_PERFORMANCE_ALLOCATION.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "  ACTIONS: SEND RADAR ORDER
```

```
            INFORMATION: RADAR ORDER, ELEVATION OF RADAR ORDER".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  FORM_T1_T2
            ALPHA:  FORM_T3
            ALPHA:  INITIALIZE_SKED_R
            ALPHA:  LOW_ELEVATION_DETERMINATION
            ALPHA:  LOW_TERMINATION
            DATA:  LOW_ELEVATION
            ENTITY_CLASS:  IMAGE
            ENTITY_CLASS:  PULSE
            ENTITY_TYPE:  DROPPED_IMAGE
            EVENT:  SCHEDULE
            EVENT:  XRB
            R_NET:  RESPONSE_TO_RADAR
            R_NET:  SKED_R
            R_NET:  XMIT_R
            SUBNET:  FORM_FRAME.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_C_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "  ACTION: SEND RADAR ORDER, DETERMINE_IMAGE_ELEVATION
               INFORMATION: IMAGE, ELEVATION OF IMAGE, RADAR ORDER,
                            TRANSMISSION TIME OF RADAR ORDER".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  INITIALIZE_SKED_R
            ALPHA:  LOW_ELEVATION_DETERMINATION
            ALPHA:  LOW_TERMINATION
            ALPHA:  UPDATE_STATE
            DATA:  DROP_REASON
            DATA:  REASON_FOR_DROP
            ENTITY_CLASS:  IMAGE
            ENTITY_CLASS:  PULSE
            ENTITY_TYPE:  DROPPED_IMAGE
            EVENT:  SCHEDULE
            EVENT:  XRB
            R_NET:  RESPONSE_TO_RADAR
            R_NET:  SKED_R
            R_NET:  XMIT_R
            SUBNET:  FORM_FRAME.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
```

```
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_D_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
      "ACTION: DROP TRACK ON HANDOVER IMAGE.
       INFORMATION: DROP TRACK C2 MESSAGE, RADAR ORDER, TRANSMISSION
       TIME OF RADAR ORDER.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  INITIALIZE_SKED_R
            ALPHA:  TERM_TRACK
            DATA:  DROP_REASON
            DATA:  ENTRY_TIME
            DATA:  REASON_FOR_DROP
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  DROPPED_IMAGE
            EVENT:  SCHEDULE
            EVENT:  XRB
            MESSAGE:  TERMINATION
            R_NET:  CC_RESPONSE
            R_NET:  SKED_R
            R_NET:  XMIT_R.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_E_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "   ACTION: MAINTAIN TRACK ON IMAGE
                INFORMATION: IMAGE&ESTIMATE OF STATE<, RADAR RETURN,
                             TIME OF LAST PROCESSED RETURN.",
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
            ALPHA:  T3_MEASUREMENT_EXTRACTION
            ALPHA:  UPDATE_STATE
```

```
            DATA:  STATE
            DECISION:  SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  IMAGE_IN_TRACK
            R_NET:  RESPONSE_TO_RADAR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_A_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            DECISION:  TRACK_PERFORMANCE_ALLOCATION.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
          "   ACTION: DROP IMAGE%REDUNDANT<
                        INFORMATION: REDUNDANT IMAGE.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  REDUN_DETERMINATION
            ALPHA:  REDUN_TERMINATION
            DATA:  REDUNDANT_IMAGE
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  DROPPED_IMAGE
            R_NET:  RESPONSE_TO_RADAR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_B_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            DECISION:  TRACK_PERFORMANCE_ALLOCATION.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
          "   ACTION: DROP IMAGE%GHOST<
                        INFORMATION: GHOST IMAGE.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  GHOST_DETERMINATION
```

```
            ALPHA:  GHOST_TERMINATION
            DATA:  GHOST_IMAGE
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  DROPPED_IMAGE
            R_NET:  RESPONSE_TO_RADAR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_C_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            DECISION:  TRACK_PERFORMANCE_ALLOCATION.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
          "     ACTION:  SEND RADAR ORDER
                         INFORMATION: RADAR ORDER.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  ASSIGN_RATE
            ALPHA:  INITIALIZE_SKED_R
            ALPHA:  MAKE_ALLOCATION
            ALPHA:  MAKE_COMMAND
            ALPHA:  PICK_COMMAND
            ALPHA:  SELECT_COMMAND
            ALPHA:  SET_LAST
            ENTITY_CLASS:  PULSE
            EVENT:  SCHEDULE
            EVENT:  XRB
            R_NET:  CONTROL_RESOURCES
            R_NET:  SKED_R
            R_NET:  XMIT_R
            SUBNET:  FORM_FRAME.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_D_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            DECISION:  TRACK_PERFORMANCE_ALLOCATION.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
```

```
TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS.
     DESCRIPTION:
          "ACTION:  SEND RADAR ORDER
                    INFORMATION: RADAR ORDER.".
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     TRACES TO:
          ALPHA:  FIND_CONFLICT
          ALPHA:  FORM_T1_T2
          ALPHA:  FORM_T3
          ALPHA:  INITIALIZE_SKED_R
          ALPHA:  MAKE_COMMAND
          ALPHA:  PICK_CANDIDATES
          ALPHA:  PICK_COMMAND
          ALPHA:  SELECT_COMMAND
          MESSAGE:  T1_T2_COMMAND
          MESSAGE:  T3_COMMAND
          R_NET:  SKED_R
          R_NET:  XMIT_R.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_E_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     TRACES TO:
          ALPHA:  REMEMBER_STOP
          MESSAGE:  T1_T2_RETURN
          MESSAGE:  T3_RETURN
          R_NET:  RESPONSE_TO_RADAR
          SUBNET:  MISSING_RETURNS.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_F_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS.
     IMPLEMENTS:
```

```
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  GHOST_DETERMINATION
            ALPHA:  LOW_ELEVATION_DETERMINATION
            ALPHA:  REDUN_DETERMINATION
            ALPHA:  TERM_TRACK
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  IMAGE_IN_TRACK.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_G_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "ACTION:  MAINTAIN ESTIMATE OF RADAR RESOURCES
            INFORMATION:  RADAR RESOURCE USAGE ESTIMATE, RADAR
            ENERGY ESTIMATE, UPPER BOUND ESTIMATE.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  DROP_PULSE
            ALPHA:  SET_COUNTED
            ALPHA:  SET_SUMMED
            ALPHA:  SUM_ENERGY
            ALPHA:  SUM_USAGE
            ENTITY_CLASS:  PULSE
            R_NET:  CONTROL_RESOURCES
            R_NET:  RADAR_SUMMARY
            SUBNET:  SUM_RETURNS
            SUBNET:  TALLY_RETURNS.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_A_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "ACTION:  ALLOCATE RADAR ORDERS
            INFORMATION:  RADAR ORDERS, IMAGE.".
```

```
      IMPLEMENTS:
           VERSION:  TLS_DPSPR.
      TRACES TO:
           ALPHA:  ASSIGN_RATE
           ALPHA:  CREATE_STATE_FILE
           ALPHA:  DROP_LAST
           ALPHA:  MAKE_ALLOCATION
           DECISION:  RADAR_SCHEDULER_PRIORITIZATION
           ENTITY_CLASS:  PULSE
           R_NET:  CONTROL_RESOURCES.
      DOCUMENTED BY:
           SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
      DESCRIPTION:
      "DPSPR PARAGRAPH 3.2.4(B), RESOURCE CONTROL, STATES THAT
       ('THE DPS SHALL ALLOCATE RADAR COMMANDS SO THAT NOT MORE
       THAN(TBD) JOULES ARE COMMANDED PER IMAGE, NOR MORE THAN
       (TBD) KILOWATTS OR (TBD) PULSES/SECOND FOR ALL IMAGES IN
       TRACK.')".
      IMPLEMENTS:
           VERSION:  TLS_DPSPR.
      TRACES TO:
           DECISION:  RADAR_RESOURCE_CONTROL_B1
           DECISION:  RADAR_RESOURCE_CONTROL_B2
           PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE
           PERFORMANCE_REQUIREMENT:  PULSES_PER_SECOND
           PERFORMANCE_REQUIREMENT:  RADIATED_POWER
           VALIDATION_PATH:  C2_HANDOVER_COMMAND_INPUT
           VALIDATION_PATH:  RADAR_COMMAND_OUTPUT
           VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES
           VALIDATION_POINT:  C2_IMAGE_HANDOVER
           VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT
           VALIDATION_POINT:  STARTING_POINT.
      DOCUMENTED BY:
           SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "ACTION: OUTPUT TO PERMANENT FILE
             INFORMATION:  TIME OF APPEARANCE OF C2 MESSAGE,
             HANDOVER IMAGE (ESTIMATED STATE).".
      IMPLEMENTS:
           VERSION:  TLS_DPSPR.
      TRACES TO:
           ALPHA:  TRACK_INITIATE
           MESSAGE:  TRACK_INITIATION
           R_NET:  CC_RESPONSE.
      DOCUMENTED BY:
           SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
```

```
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_A_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
     DESCRIPTION:
               "ACTION:  OUTPUT TO PERMANENT FILE
               INFORMATION:  TIME OF DROP TRACK, REASON FOR
             DROP TRACK,".
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     TRACES TO:
          ALPHA:  GHOST_DETERMINATION
          ALPHA:  GHOST_TERMINATION
          ALPHA:  LOW_ELEVATION_DETERMINATION
          ALPHA:  LOW_TERMINATION
          ALPHA:  REGUN_DETERMINATION
          ALPHA:  REGUN_TERMINATION
          ALPHA:  SET_DROP
          ALPHA:  TERM_TRACK
          ENTITY_TYPE:  DROPPED_IMAGE
          MESSAGE:  TRACK_TERMINATION
          R_NET:  CC_RESPONSE
          R_NET:  RESPONSE_TO_RADAR
          SUBNET:  RECORD_DROP.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_B_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS.
     DESCRIPTION:
               "ACTION:  OUTPUT TO PERMANENT FILE, UPDATE STATE
                INFORMATION: IMAGE STATE ESTIMATE.".
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     TRACES TO:
          ALPHA:  FORM_UPDATE
          ALPHA:  SET_PULSE
          ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
          ALPHA:  T3_MEASUREMENT_EXTRACTION
          ALPHA:  UPDATE_STATE
          ENTITY_CLASS:  PULSE
          ENTITY_TYPE:  RETURNED_PULSE
```

```
            MESSAGE:  STATE_UPDATE
            R_NET:  RESPONSE_TO_RADAR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_C_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      DESCRIPTION:
            "ACTION:  OUTPUT TO PERMANENT FILE
            INFORMATION:  RADAR RESOURCE USAGE.".
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            ALPHA:  COMPLETE_SUMMARY
            ALPHA:  DROP_PULSE
            ALPHA:  SET_SUMMED
            ALPHA:  SUM_USAGE
            ENTITY_CLASS:  PULSE
            ENTITY_TYPE:  RETURNED_PULSE
            EVENT:  SUMMARIZE
            MESSAGE:  RADAR_USAGE
            R_NET:  RADAR_SUMMARY
            SUBNET:  SUM_RETURNS.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_D_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      TRACES TO:
            SUBSYSTEM:  SSC2
            SUBSYSTEM:  SSPERMEL
            SUBSYSTEM:  SSRADAR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
```

```
TLS_DPSPR_SECTION_3_0_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_1_FUNCTIONAL_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     TRACES TO:
          SUBSYSTEM:  SSRADAR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_1_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     TRACES TO:
          SUBSYSTEM:  SSRADAR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     INCORPORATES:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
     TRACES TO:
          INPUT_INTERFACE:  RADAR_IN
          OUTPUT_INTERFACE:  RADAR_OUT.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     INCORPORATES:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_1_PERFORMANCE_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_2_PERFORMANCE_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_3_PERFORMANCE_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_4_PERFORMANCE_REQUIREMENTS
```

```
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_5_PERFORMANCE_REQUIREMENTS.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      INCORPORATES:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_B_FUNCTIONAL_REQUIREMENTS.
      TRACES TO:
            INPUT_INTERFACE:  CC_IN
            OUTPUT_INTERFACE:  CC_OUT
            R_NET:  CC_RESPONSE
            SUBSYSTEM:  SSC2.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_1_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      INCORPORATES:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS.
      TRACES TO:
            OUTPUT_INTERFACE:  RADAR_OUT
            R_NET:  RESPONSE_TO_RADAR
            R_NET:  SKED_R
            R_NET:  XMIT_R.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
```

```
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_2_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      INCORPORATES:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS.
      TRACES TO:
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  IMAGE_IN_TRACK
            INPUT_INTERFACE:  RADAR_IN
            MESSAGE:  T1_T2_COMMAND
            MESSAGE:  T1_T2_RETURN
            MESSAGE:  T3_COMMAND
            MESSAGE:  T3_RETURN
            OUTPUT_INTERFACE:  RADAR_OUT
            R_NET:  RESPONSE_TO_RADAR
            R_NET:  SKED_R
            R_NET:  XMIT_R.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_3_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
```

```
            TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      INCORPORATES:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS.
      TRACES TO:
            R_NET:  CONTROL_RESOURCES
            R_NET:  RADAR_SUMMARY
            SUBNET:  SUM_RETURNS
            SUBNET:  TALLY_RETURNS.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_4_PERFORMANCE_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
      IMPLEMENTS:
            VERSION:  TLS_DPSPR.
      INCORPORATES:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS.
      TRACES TO:
            OUTPUT_INTERFACE:  DATA_RECORD
            R_NET:  CC_RESPONSE
            R_NET:  RADAR_SUMMARY
            R_NET:  RESPONSE_TO_RADAR
            SUBSYSTEM:  SSPERMFL.
      DOCUMENTED BY:
            SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
      INCORPORATED IN:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
```

```
TLS_DPSPR_SUBSECTION_3_2_5_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
     INCORPORATED IN:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS.

ORIGINATING_REQUIREMENT:
TLS_RADAR_DPS_IFS_PARAGRAPH_3_2_9_FUNCTIONAL_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     TRACES TO:
          DATA:  RADAR_CLOCK_TIME.
     DOCUMENTED BY:
          SOURCE:  TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

ORIGINATING_REQUIREMENT:
TLS_RADAR_DPS_IFS_PARAGRAPH_3_2_9_PERFORMANCE_REQUIREMENTS.
     IMPLEMENTS:
          VERSION:  TLS_DPSPR.
     DOCUMENTED BY:
          SOURCE:  TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

OUTPUT_INTERFACE:  CC_OUT.
     CONNECTS TO:
          SUBSYSTEM:  SSC2.
     PASSES:
          MESSAGE:  ACKNOWLEDGEMENT.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  CC_RESPONSE.

OUTPUT_INTERFACE:  DATA_RECORD.
     CONNECTS TO:
          SUBSYSTEM:  SSPERMFL.
     PASSES:
          MESSAGE:  RADAR_USAGE
          MESSAGE:  STATE_UPDATE
          MESSAGE:  TRACK_INITIATION
          MESSAGE:  TRACK_TERMINATION.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  CC_RESPONSE
          R_NET:  RADAR_SUMMARY
          R_NET:  RESPONSE_TO_RADAR.

OUTPUT_INTERFACE:  RADAR_OUT.
     DESCRIPTION:
          "THE RADOUT INTERFACE PROVIDES THE MECHANISM THROUGH
           WHICH THE DPS ISSUES COMMANDS TO THE RADAR SUBSYSTEM.
```

```
            THE RADAR SUBSYSTEM WILL EXECUTE ONLY THE COMMANDS
            ISSUED BY THE DPS AND WILL TRANSMIT ONE PULSE FOR EACH
            COMMAND WHICH SATISFIES THE RADAR SUBSYSTEM AND
            INTERFACE CONSTRAINTS. THE RADAR SUBSYSTEM WILL EXECUTE
            THE COMMANDS IN THE ORDER RECEIVED AND WILL BEGIN
            EXECUTION AFTER RECEIPT OF END_OF_TRANSMISSION.",
      ENTERED_BY: "H.A.HELTON, APR. 30, 1976.",
      CONNECTS TO:
           SUBSYSTEM:  SSRADAR.
      IMPLEMENTS:
           VERSION:  ORIGINAL_PUBLICATION_DATED_AUGUST_1975.
      PASSES:
           MESSAGE:  T1_T2_COMMAND
           MESSAGE:  T3_COMMAND.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
           R_NET:  XMIT_R.

PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE.
      TEST:
      "CONST
        ENERGY_LIMIT=5.0;(* TEMPORARY REPLACEMENT FOR (TBD). *)
       VAR
        IMAGE_ENERGY: REAL;
       BEGIN
        ENERGY_PER_IMAGE:=TRUE;
        RETRIEVE FIRST RECORDING FOR STARTING_POINT;
        FOR EACH C2_IMAGE_HANDOVER RECORDING
        DO
          IMAGE_ENERGY:=0.0;
          FOR EACH RADAR_COMMAND_OUTPUT_POINT RECORDING
            SUCH THAT (RADAR_COMMAND_OUTPUT_POINT.TARGET_ID=
                       C2_IMAGE_HANDOVER.HB_ID)
          DO
            SELECT FIRST RECORD FROM STARTING_POINT.WAVEFORM_TABLE
              SUCH THAT (RADAR_COMMAND_OUTPUT_POINT.RADAR_TYPE=
                         STARTING_POINT.WF_NAME);
              IF RECORD_FOUND THEN
                 IMAGE_ENERGY:=IMAGE_ENERGY+
                               STARTING_POINT.WF_CHARACTERISTICS;
          ENDFOREACH;
          IF (IMAGE_ENERGY>ENERGY_LIMIT) THEN
            ENERGY_PER_IMAGE:=FALSE;
        ENDFOREACH
       END;",
      CONSTRAINS:
           VALIDATION_PATH:  C2_HANDOVER_COMMAND_INPUT
           VALIDATION_PATH:  RADAR_COMMAND_OUTPUT
           VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES.
      TRACED FROM:
```

```
            DECISION:  RADAR_RESOURCE_CONTROL_R1
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.

PERFORMANCE_REQUIREMENT:  PULSES_PER_SECOND.
      TEST:
      "CONST
       PULSE_RATE_LIMIT=20.0; (* TEMPORARY REPLACEMENT FOR (TBD). *)
       VAR
       PULSE_RATE: INTEGER;
       INTERVAL: REAL;
       BEGIN
        PULSES_PER_SECOND:=TRUE;
        FOR EACH RADAR_COMMAND_OUTPUT_POINT RECORDING
        DO
          PULSE_RATE:=0;
          INTERVAL:=RADAR_COMMAND_OUTPUT_POINT.TRANSMIT_START;
          FOR EACH RADAR_COMMAND_OUTPUT_POINT RECORDING
            SUCH THAT((RADAR_COMMAND_OUTPUT_POINT.TRANSMIT_START<=
                       INTERVAL+1.0) AND
                       (RADAR_COMMAND_OUTPUT_POINT.TRANSMIT_START>=
                       INTERVAL))
          DO
            PULSE_RATE:=PULSE_RATE+1;
          ENDFOREACH;
          IF (PULSE_RATE>PULSE_RATE_LIMIT) THEN
            PULSES_PER_SECOND:=FALSE;
        ENDFOREACH
       END;",
      CONSTRAINS:
           VALIDATION_PATH:  RADAR_COMMAND_OUTPUT.
      TRACED FROM:
           ORIGINATING_REQUIREMENT:
           TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.

PERFORMANCE_REQUIREMENT:  RADIATED_POWER.
      TEST:
      "CONST
       RADIATED_POWER_LIMIT=5.0; (* TEMPORARY REPLACEMENT FOR (TBD) *)
       VAR
       RADAR_POWER: REAL;
       INTERVAL: REAL;
       BEGIN
        RADIATED_POWER:=TRUE;
        RETRIEVE FIRST RECORDING FOR STARTING_POINT;
        FOR EACH RADAR_COMMAND_OUTPUT_POINT RECORDING
        DO
          RADAR_POWER:=1.0;
          INTERVAL:=RADAR_COMMAND_OUTPUT_POINT.TRANSMIT_START;
          FOR EACH RADAR_COMMAND_OUTPUT_POINT RECORDING
            SUCH THAT ((RADAR_COMMAND_OUTPUT_POINT.TRANSMIT_START<=
                        INTERVAL+1.0) AND
                       (RADAR_COMMAND_OUTPUT_POINT.TRANSMIT_START>=
                        INTERVAL))
          DO
            SELECT FIRST RECORD FROM STARTING_POINT.WAVEFORM_TABLE
              SUCH THAT (RADAR_COMMAND_OUTPUT_POINT.RADAR_TYPE=
                         STARTING_POINT.WF_NAME);
```

```
            IF RECORD_FOUND THEN
               RADAR_POWER:=RADAR_POWER+
                           STARTING_POINT.WF_CHARACTERISTICS;
         ENDFOREACH;
         IF (RADAR_POWER>RADIATED_POWER_LIMIT) THEN
           RADIATED_POWER:=FALSE;
       ENDFOREACH
      END;".
    CONSTRAINS:
         VALIDATION_PATH:  RADAR_COMMAND_OUTPUT
         VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES.
    TRACED FROM:
         DECISION:  RADAR_RESOURCE_CONTROL_B2
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.

R_NET:  CC_RESPONSE.
    REFERS TO:
         ALPHA:  ACKNOWLEDGE
         ALPHA:  CC_ERROR_PROCESSING
         ALPHA:  ENGAGEMENT_INITIATION
         ALPHA:  STARTER
         ALPHA:  TERM_ENGAGEMENT
         ALPHA:  TERM_TRACK
         ALPHA:  TRACK_INITIATE
         ALPHA:  VALIDATE_HEADER
         DATA:  COMMAND_ID
         DATA:  FOUND
         DATA:  MO_ID
         DATA:  IMAGE_ID
         ENTITY_CLASS:  IMAGE
         EVENT:  ALLOCATE
         EVENT:  SCHEDULE
         EVENT:  SUMMARIZE
         INPUT_INTERFACE:  CC_IN
         OUTPUT_INTERFACE:  CC_OUT
         OUTPUT_INTERFACE:  DATA_RECORD
         SUBNET:  RECORD_DRAF
         VALIDATION_POINT:  C2_IMAGE_HANDOVER
         VALIDATION_POINT:  STARTING_POINT.
    ENABLED BY:
         INPUT_INTERFACE:  CC_IN.
    TRACED FROM:
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
    STRUCTURE:
       INPUT_INTERFACE:  CC_IN
       ALPHA:  VALIDATE_HEADER
```

```
        DO
            ALPHA:  ACKNOWLEDGE
            OUTPUT_INTERFACE:  CC_OUT
        AND
            CONSIDER  DATA:  COMMAND_ID
            IF (HANDOVER_IMAGE)
                ALPHA:  TRACK_INITIATE
                VALIDATION_POINT:  C2_IMAGE_HANDOVER
                EVENT:  ALLOCATE
                OUTPUT_INTERFACE:  DATA_RECORD
            OR (INITIATE_ENGAGEMENT_MODE)
                ALPHA:  STARTER
                VALIDATION_POINT:  STARTING_POINT
                ALPHA:  ENGAGEMENT_INITIATION
                EVENT:  SCHEDULE
                EVENT:  SUMMARIZE
                TERMINATE
            OR (TERMINATE_ENGAGEMENT_MODE)
                ALPHA:  TERM_ENGAGEMENT
                TERMINATE
            OR (DROP_TRACK)
                SELECT ENTITY_CLASS:  IMAGE SUCH THAT (IMAGE_ID=HO_ID)
                IF (FOUND)
                    SUBNET:  RECORD_DROP
                    ALPHA:  TERM_TRACK
                    OUTPUT_INTERFACE:  DATA_RECORD
                OTHERWISE
                    ALPHA:  CC_ERROR_PROCESSING
                    TERMINATE
                END
            OR (CC_MESSAGE_ERROR)
                ALPHA:  CC_ERROR_PROCESSING
                TERMINATE
            END
        END.


R_NET:  CONTROL_RESOURCES.
        DESCRIPTION:
            "THE TLS_DPS SHALL IMPLEMENT THE REQUIREMENTS SPECIFIED
             IN THE DPSPR, REFERENCE 2.2, ASSOCIATED WITH MANAGEMENT
             AND CONTROL OF TLS RESOURCES AND SHALL PERFORM THE
             FUNCTIONS HEREIN DEFINED AND DIAGRAMMED IN THE CONTRES
             R_NET.
               THE DPS SHALL ASSIGN TRACK RATES AND ENERGY ALLOWANCES
                 TO EACH IMAGE HANDED OVER FROM COMMAND AND CONTROL
                 AND SHALL DETERMINE ENERGY BOUNDS AND TRACK RATE
                 ENVELOPES FOR EACH HANDOVER IMAGE WHICH REMAINS IN
                 TRACK STATUS. THE ENERGY BOUNDS AND TRACK RATE
                 ENVELOPES SHALL BE MAINTAINED WITHIN AN ACCURACY
                 AND RESOLUTION TOLERANCE SUFFICIENT TO MEET THE
                 SPECIFIED REQUIREMENTS FOR DETERMINATION OF TARGET
                 INTERCEPT CONDITIONS.
               THE DPS SHALL ASSESS STATUS OF THE TLS RESOURCES AND
                 SHALL ALLOCATE TLS RESOURCES TO EACH HANDOVER IMAGE
                 BASED ON RADAR SUBSYSTEM PERFORMANCE CAPABILITIES
                 AND SHALL MAINTAIN A GRACEFUL DEGRADATION POSTURE
```

```
                WHILE UNDER OVERLOAD CONDITIONS,
              THE DPS SHALL GENERATE TLS RESOURCE UTILIZATION
                PROFILES AND SHALL COMMIT THESE DATA TO PERMANENT
                FILE THROUGH THE DATA RECORD OUTPUT INTERFACE.",
        REFERS TO:
              ALPHA:  ASSIGN_RATE
              ALPHA:  CREATE_STATE_FILE
              ALPHA:  DROP_LOST
              ALPHA:  MAKE_ALLOCATION
              ENTITY_TYPE:  IMAGE_IN_TRACK
              ENTITY_TYPE:  LOST_PULSE
              ENTITY_TYPE:  RETURNED_PULSE
              SUBNET:  TALLY_RETURNS,
        ENABLED BY:
              EVENT:  ALLOCATE,
        TRACED FROM:
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS,
        STRUCTURE:
           DO
              FOR EACH  ENTITY_TYPE:  RETURNED_PULSE
                 DO SUBNET:  TALLY_RETURNS END
           AND
              FOR EACH  ENTITY_TYPE:  LOST_PULSE
                 DO ALPHA:  DROP_LOST END
           AND
              FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK
                 DO ALPHA:  CREATE_STATE_FILE END
           END
           ALPHA:  MAKE_ALLOCATION
           FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK
              DO ALPHA:  ASSIGN_RATE END
           TERMINATE
        END,


R_NET:  RADAR_SUMMARY,
        REFERS TO:
              ALPHA:  COMPLETE_SUMMARY
              DATA:  MODE
              ENTITY_TYPE:  RETURNED_PULSE
              EVENT:  SUMMARIZE
              OUTPUT_INTERFACE:  DATA_RECORD
              SUBNET:  SUM_RETURNS,
        ENABLED BY:
              EVENT:  SUMMARIZE,
        TRACED FROM:
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
              TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS
              ORIGINATING_REQUIREMENT:
```

```
        TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
    STRUCTURE:
       CONSIDER DATA: MODE
       IF (ENGAGED)
          FOR EACH ENTITY_TYPE: RETURNED_PULSE
             DO SUBNET: SUM_RETURNS END
          ALPHA: COMPLETE_SUMMARY
          EVENT: SUMMARIZE
          OUTPUT_INTERFACE: DATA_RECORD
       OF (STANDBY)
          TERMINATE
       END
    END.


R_NET: RADAR_TIMING.
    DESCRIPTION:
          "RADAR_TIMING MAINTAINS A RECORD OF THE
           RADAR_CLOCK_TIME.".
    REFERS TO:
         ALPHA: UPDATE_RADAR_CLOCK
         INPUT_INTERFACE: RADAR_CLOCK_IN.
    ENABLED BY:
         INPUT_INTERFACE: RADAR_CLOCK_IN.
    STRUCTURE:
       INPUT_INTERFACE: RADAR_CLOCK_IN
       ALPHA: UPDATE_RADAR_CLOCK
       TERMINATE
    END.


R_NET: RESPONSE_TO_RADAR.
    DESCRIPTION:
          "THE TLS_DPS SHALL IMPLEMENT THE REQUIREMENTS SPECIFIED
           IN THE TLS_RADAR_DPS_INTERFACE_SPECIFICATION ASSOCIATED
           WITH PROCESSING RADAR SUBSYSTEM RESPONSES TO COMMANDS,
           AND SHALL PERFORM THE FUNCTIONS HEREIN DEFINED AND
           DIAGRAMMED IN THE RESPRAD R_NET.
             THE DPS SHALL RECEIVE AND PROCESS RADAR MESSAGES
               TRANSMITTED BY THE TLS RADAR SUBSYSTEM AND SHALL
               INTERROGATE EACH MESSAGE FOR ERRORS AND SHALL
               PROCESS ALL DETECTED MESSAGE ERRORS.
             THE DPS SHALL, UPON RECEIPT OF ERROR_FREE RADAR
               MESSAGES, DETECT AND PROCESS REDUNDANT_IMAGES,
               GHOST_IMAGES, AND LOW_ELEVATION_IMAGES, AND SHALL
               UPDATE STATE_PARAMETERS FOR EACH IMAGE_IN_TRACK.
             THE DPS SHALL TERMINATE TRACK ON EACH IMAGE WHICH IS
               DETERMINED TO BE EITHER A REDUNDANT OR GHOST IMAGE
               OR WHICH IS FOUND TO EXCEED THE LOW_ELEVATION
               CONSTRAINTS AND SHALL MAINTAIN TRACK ON EACH IMAGE
               DETERMINED TO BE REAL.
             THE DPS SHALL CONSTRUCT AND MAINTAIN DESCRIPTIVE DATA
               FILES FOR EACH IMAGE WHICH IS EITHER MAINTAINED IN
               TRACK OR DROPPED FROM TRACK AND SHALL PERIODICALLY
               COMMIT THESE DATA TO PERMANENT RECORDS THROUGH THE
               DATA RECORDS INTERFACE.".
```

```
IMPLEMENTS:
     VERSION:  PDL2_COMPILER_LIMITATION_COMPENSATION.
REFERS TO:
     ALPHA:  FORM_UPDATE
     ALPHA:  GHOST_DETERMINATION
     ALPHA:  GHOST_TERMINATION
     ALPHA:  LOW_ELEVATION_DETERMINATION
     ALPHA:  LOW_TERMINATION
     ALPHA:  REDUN_DETERMINATION
     ALPHA:  REDUN_TERMINATION
     ALPHA:  REMEMBER_STOP
     ALPHA:  RR_ERROR_PROCESSING
     ALPHA:  UPDATE_STATE
     DATA:  FOUND
     DATA:  GHOST_IMAGE
     DATA:  IMAGE_ID
     DATA:  LOW_ELEVATION
     DATA:  PULSE_ID
     DATA:  PULSE_TYPE
     DATA:  RADAR_TYPE
     DATA:  REDUNDANT_IMAGE
     DATA:  RR_ORDER_ID
     DATA:  TARGET_ID
     DATA:  VALID_RETURN
     ENTITY_CLASS:  IMAGE
     ENTITY_CLASS:  PULSE
     ENTITY_TYPE:  IMAGE_IN_TRACK
     INPUT_INTERFACE:  RADAR_IN
     OUTPUT_INTERFACE:  DATA_RECORD
     SUBNET:  EXTRACT_MEASUREMENT
     SUBNET:  MISSING_RETURNS
     SUBNET:  RECORD_DROP.
ENABLED BY:
     INPUT_INTERFACE:  RADAR_IN.
TRACED FROM:
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
     TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
     ORIGINATING_REQUIREMENT:
```

```
      TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS
      ORIGINATING_REQUIREMENT:
      TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.
STRUCTURE:
   INPUT_INTERFACE:  RADAR_IN
   SELECT ENTITY_CLASS:  PULSE SUCH THAT (PULSE_ID=RR_ORDER_ID)
   IF (FOUND)
      IF (RADAR_TYPE=PULSE_TYPE)
         DO
            ALPHA:  REMEMBER_STOP
            FOR EACH  ENTITY_CLASS:  PULSE
               DO SUBNET:  MISSING_RETURNS END
            SELECT ENTITY_CLASS:  PULSE SUCH THAT
            (PULSE_ID=RR_ORDER_ID)
            TERMINATE
         AND
            SUBNET:  EXTRACT_MEASUREMENT
            IF (VALID_RETURN)
               ALPHA:  UPDATE_STATE
               DO
                  ALPHA:  LOW_ELEVATION_DETERMINATION
                  IF (LOW_ELEVATION)
                     SUBNET:  RECORD_DROP
                     ALPHA:  LOW_TERMINATION
                     OUTPUT_INTERFACE:  DATA_RECORD
                  OTHERWISE
                     TERMINATE
                  END
               AND
                  ALPHA:  FARM_UPDATE
                  OUTPUT_INTERFACE:  DATA_RECORD
               AND
                  FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK
                  SUCH THAT (IMAGE_ID<>TARGET_ID)
                     DO ALPHA:  REDUN_DETERMINATION END
                  SELECT ENTITY_CLASS:  IMAGE SUCH THAT
                  (IMAGE_ID=TARGET_ID)
                  IF (REDUNDANT_IMAGE)
                     SUBNET:  RECORD_DROP
                     ALPHA:  REDUN_TERMINATION
                     OUTPUT_INTERFACE:  DATA_RECORD
                  OTHERWISE
                     TERMINATE
                  END
               END
            OTHERWISE
               ALPHA:  GHOST_DETERMINATION
               IF (GHOST_IMAGE)
                  SUBNET:  RECORD_DROP
                  ALPHA:  GHOST_TERMINATION
                  OUTPUT_INTERFACE:  DATA_RECORD
               OTHERWISE
                  TERMINATE
               END
            END
         END
      OTHERWISE
         ALPHA:  RR_ERROR_PROCESSING
```

```
            TERMINATE
         END
      OTHERWISE
         ALPHA:  RR_ERROR_PROCESSING
         TERMINATE
      END
   END.


R_NET:  SKED_R,
   DESCRIPTION:
         "SKED_R CONSTRUCTS THE ORDERED FILE OF DATA FOR
          A FRAME.",
   REFERS TO:
         ALPHA:  INITIALIZE_SKED_R
         ALPHA:  PICK_CANDIDATES
         DATA:  LAST_PULSE
         DATA:  MODE
         DATA:  TEOF
         DATA:  TRACK_RATE
         ENTITY_TYPE:  IMAGE_IN_TRACK
         EVENT:  XRB
         FILE:  CANDIDATE
         SUBNET:  FORM_FRAME,
   ENABLED BY:
         EVENT:  SCHEDULE,
   TRACED FROM:
         DECISION:  RADAR_SCHEDULER_PRIORITIZATION
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
         ORIGINATING_REQUIREMENT:
         TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS,
   STRUCTURE:
      CONSIDER  DATA:  MODE
      IF (ENGAGED)
         ALPHA:  INITIALIZE_SKED_R
         FOR EACH  ENTITY_TYPE:  IMAGE_IN_TRACK SUCH THAT
         (LAST_PULSE+(1.0/TRACK_RATE)<TEOF)
            DO ALPHA:  PICK_CANDIDATES END
         FOR EACH  FILE:  CANDIDATE RECORD
            DO SUBNET:  FORM_FRAME END
         EVENT:  XRB
         TERMINATE
      OR (STANDBY)
```

```
            TERMINATE
         END
      END.


R_NET:  XMIT_R.
      DESCRIPTION:
            "XMIT_R BUILDS AND FORWARDS TO THE OUTPUT_INTERFACE
             RADAR_OUT THE COMMANDS OF THE FRAME,".
      REFERS TO:
            ALPHA:  FORM_T1_T2
            ALPHA:  FORM_T3
            ALPHA:  PICK_COMMAND
            ALPHA:  SELECT_COMMAND
            DATA:  RADAR_TYPE
            DATA:  RECORD_FOUND
            EVENT:  SCHEDULE
            EVENT:  XRB
            OUTPUT_INTERFACE:  RADAR_OUT
            VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
      ENABLED BY:
            EVENT:  XRB.
      TRACED FROM:
            DECISION:  RADAR_SCHEDULER_PRIORITIZATION
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS.
      STRUCTURE:
         ALPHA:  SELECT_COMMAND
         IF (RECORD_FOUND)
            ALPHA:  PICK_COMMAND
            EVENT:  XRB
            CONSIDER  DATA:  RADAR_TYPE
            IF (T3)
               ALPHA:  FORM_T3
            OR (T1 OR T2)
               ALPHA:  FORM_T1_T2
            END
            VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT
            OUTPUT_INTERFACE:  RADAR_OUT
         OTHERWISE
            EVENT:  SCHEDULE
            TERMINATE
```

```
    END
END.

SOURCE:  TLS_DPSPR_X1_TRACK_LOOP_EXPERIMENT.
    DOCUMENTS:
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_1_A_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_1_B_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_1_B_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_A_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_C_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_D_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_2_E_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_A_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_B_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_C_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_D_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_E_PERFORMANCE_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
        ORIGINATING_REQUIREMENT:
        TLS_DPSPR_PARAGRAPH_3_2_3_F_PERFORMANCE_REQUIREMENTS
```

```
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_G_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_A_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_A_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_B_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_C_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_D_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_0_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_1_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_1_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_1_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_2_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_3_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_4_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS
```

```
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_SUBSECTION_3_2_5_PERFORMANCE_REQUIREMENTS.

      SOURCE:  TLS_RADAR_DPS_INTERFACE_SPECIFICATION.
            DOCUMENTS:
            ORIGINATING_REQUIREMENT:
            TLS_RADAR_DPS_IFS_PARAGRAPH_3_2_9_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_RADAR_DPS_IFS_PARAGRAPH_3_2_9_PERFORMANCE_REQUIREMENTS.

      SUBNET:  EXTRACT_MEASUREMENT.
            IMPLEMENTS:
            VERSION:  PDL2_COMPILER_LIMITATION_COMPENSATION.
            REFERS TO:
            ALPHA:  SET_PULSE
            ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
            ALPHA:  T3_MEASUREMENT_EXTRACTION
            DATA:  IMAGE_ID
            DATA:  TARGET_ID
            ENTITY_CLASS:  IMAGE
            ENTITY_CLASS:  PULSE
            ENTITY_TYPE:  LAST_PULSE
            ENTITY_TYPE:  RETURNED_PULSE
            ENTITY_TYPE:  T1_T2_PULSE
            ENTITY_TYPE:  T3_PULSE.
            REFERRED BY:
            R_NET:  RESPONSE_TO_RADAR.
            STRUCTURE:
               SELECT ENTITY_CLASS:  IMAGE SUCH THAT (IMAGE_ID=TARGET_ID)
               CONSIDER  ENTITY_CLASS:  PULSE
               IF (T3_PULSE)
                  ALPHA:  T3_MEASUREMENT_EXTRACTION
               OR (T1_T2_PULSE)
                  ALPHA:  T1_T2_MEASUREMENT_EXTRACTION
               OR (LAST_PULSE OR RETURNED_PULSE)
               END
               ALPHA:  SET_PULSE
               RETURN
            END.


      SUBNET:  FORM_FRAME.
            DESCRIPTION: "FORM_FRAME PROVIDES RADAR CONFLICT RESOLUTION.".
            REFERS TO:
            ALPHA:  FIND_CONFLICT
            ALPHA:  MAKE_COMMAND
            ALPHA:  SET_LAST
            DATA:  CANDIDATE_IMAGE_ID
            DATA:  DROP_FLAG
            DATA:  IMAGE_ID
            ENTITY_TYPE:  IMAGE_IN_TRACK.
            TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
```

```
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  SKED_R.
      STRUCTURE:
         ALPHA:  FIND_CONFLICT
         IF (NOT DROP_FLAG)
            ALPHA:  MAKE_COMMAND
            SELECT ENTITY_TYPE:  IMAGE_IN_TRACK SUCH THAT
            (IMAGE_ID=CANDIDATE_IMAGE_ID)
            (*MUST SUCCEED SINCE SELECTED ON CALLING NET*)
            ALPHA:  SET_LAST
         OTHERWISE
         END
         RETURN
      END.


SUBNET:  MISSING_RETURNS.
      REFERS TO:
            ALPHA:  SET_LAST
            DATA:  RECEIVE_STOP
            DATA:  STOP_TIME
            ENTITY_CLASS:  PULSE
            ENTITY_TYPE:  LOST_PULSE
            ENTITY_TYPE:  RETURNED_PULSE
            ENTITY_TYPE:  T1_T2_PULSE
            ENTITY_TYPE:  T3_PULSE.
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS.
      REFERRED BY:
            R_NET:  RESPONSE_TO_RADAR.
      STRUCTURE:
         CONSIDER  ENTITY_CLASS:  PULSE
         IF (T1_T2_PULSE)
            IF (RECEIVE_STOP<STOP_TIME)
               ALPHA:  SET_LAST
            OTHERWISE
            END
         OR (T3_PULSE)
            IF (RECEIVE_STOP<STOP_TIME)
               ALPHA:  SET_LAST
            OTHERWISE
            END
         OR (LOST_PULSE OR RETURNED_PULSE)
         END
         RETURN
      END.


SUBNET:  RECORD_DROP.
      REFERS TO:
            ALPHA:  SET_DROP
            ENTITY_CLASS:  IMAGE
            ENTITY_TYPE:  DROPPED_IMAGE
```

```
          ENTITY_TYPE:  IMAGE_IN_TRACK
          EVENT:  ALLOCATE.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  CC_RESPONSE
          R_NET:  RESPONSE_TO_RADAR.
     STRUCTURE:
        CONSIDER  ENTITY_CLASS:  IMAGE
        IF (IMAGE_IN_TRACK)
           ALPHA:  SET_DROP
           EVENT:  ALLOCATE
        OR (DROPPED_IMAGE)
        END
        RETURN
     END.


SUBNET:  SUM_RETURNS.
     REFERS TO:
          ALPHA:  DROP_PULSE
          ALPHA:  SET_SUMMED
          ALPHA:  SUM_USAGE
          DATA:  ACCOUNTED_FOR.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  RADAR_SUMMARY.
     STRUCTURE:
        CONSIDER  DATA:  ACCOUNTED_FOR
        IF (COUNTED)
           ALPHA:  SUM_USAGE
           ALPHA:  DROP_PULSE
        OR (NEITHER)
           ALPHA:  SUM_USAGE
           ALPHA:  SET_SUMMED
        OR (SUMMED)
        END
        RETURN
     END.


SUBNET:  TALLY_RETURNS.
     REFERS TO:
          ALPHA:  DROP_PULSE
          ALPHA:  SET_COUNTED
          ALPHA:  SUM_ENERGY
          DATA:  ACCOUNTED_FOR.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
```

```
          TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS.
     REFERRED BY:
          R_NET:  CONTROL_RESOURCES.
     STRUCTURE:
        CONSIDER  DATA:  ACCOUNTED_FOR
        IF (SUMMED)
           ALPHA:  SUM_ENERGY
           ALPHA:  DROP_PULSE
        OR (NEITHER)
           ALPHA:  SUM_ENERGY
           ALPHA:  SET_COUNTED
        OR (COUNTED)
        END
        RETURN
     END.


SUBSYSTEM:  SSC2
(*I*).
     CONNECTED TO:
          INPUT_INTERFACE:  CC_IN
          OUTPUT_INTERFACE:  CC_OUT.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS.

SUBSYSTEM:  SSPERMFL
(*K*).
     CONNECTED TO:
          OUTPUT_INTERFACE:  DATA_RECORD.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS.

SUBSYSTEM:  SSRADAR
(*J*).
     CONNECTED TO:
          INPUT_INTERFACE:  RADAR_CLOCK_IN
          INPUT_INTERFACE:  RADAR_IN
          OUTPUT_INTERFACE:  RADAR_OUT.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_1_FUNCTIONAL_REQUIREMENTS
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_SECTION_3_1_PERFORMANCE_REQUIREMENTS.

SYNONYM:  CKRADMES.

SYNONYM:  CONTRES.

SYNONYM:  RADIN.
```

```
            TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS,
      PATH :
         VALIDATION_POINT:  STARTING_POINT
      END,


VALIDATION_POINT:  C2_IMAGE_HANDOVER,
      RECORDS:
            DATA:  HO_ID,
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS,
      REFERRED BY:
            R_NET:  CC_RESPONSE
            VALIDATION_PATH:  C2_HANDOVER_COMMAND_INPUT,

VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT,
      RECORDS:
            DATA:  RADAR_TYPE
            DATA:  TARGET_ID
            DATA:  TRANSMIT_START,
      TRACED FROM:
            DECISION:  RADAR_RESOURCE_CONTROL_B1
            DECISION:  RADAR_RESOURCE_CONTROL_B2
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS,
      REFERRED BY:
            R_NET:  XMIT_R
            VALIDATION_PATH:  RADAR_COMMAND_OUTPUT,

VALIDATION_POINT:  STARTING_POINT,
      RECORDS:
            DATA:  WF_CHARACTERISTICS
            DATA:  WF_NAME
            FILE:  WAVEFORM_TABLE,
      TRACED FROM:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS,
      REFERRED BY:
            R_NET:  CC_RESPONSE
            VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES,

VERSION:  ORIGINAL_PUBLICATION_DATED_AUGUST_1975,
      IMPLEMENTED BY:
            INPUT_INTERFACE:  RADAR_IN
            OUTPUT_INTERFACE:  RADAR_OUT,

VERSION:  PDL2_COMPILER_LIMITATION_COMPENSATION,
      IMPLEMENTED BY:
            R_NET:  RESPONSE_TO_RADAR
            SUBNET:  EXTRACT_MEASUREMENT,

VERSION:  TLS_DPSPR,
      IMPLEMENTED BY:
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_A_FUNCTIONAL_REQUIREMENTS
            ORIGINATING_REQUIREMENT:
            TLS_DPSPR_PARAGRAPH_3_2_1_A_PERFORMANCE_REQUIREMENTS
```

```
SYNONYM:  RADOUT.

SYNONYM:  RESPRAD.

SYNONYM:  T1T2CMD.
     EQUATES TO:
          MESSAGE:  T1_T2_COMMAND.

SYNONYM:  T1T2RTN.
     EQUATES TO:
          MESSAGE:  T1_T2_RETURN.

SYNONYM:  T3CMD.
     EQUATES TO:
          MESSAGE:  T3_COMMAND.

SYNONYM:  T3RTN.
     EQUATES TO:
          MESSAGE:  T3_RETURN.

VALIDATION_PATH:  C2_HANDOVER_COMMAND_INPUT.
     REFERS TO:
          VALIDATION_POINT:  C2_IMAGE_HANDOVER.
     CONSTRAINED BY:
          PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     PATH :
        VALIDATION_POINT:  C2_IMAGE_HANDOVER
     END.


VALIDATION_PATH:  RADAR_COMMAND_OUTPUT.
     REFERS TO:
          VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT.
     CONSTRAINED BY:
          PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE
          PERFORMANCE_REQUIREMENT:  PULSES_PER_SECOND
          PERFORMANCE_REQUIREMENT:  RADIATED_POWER.
     TRACED FROM:
          DECISION:  RADAR_RESOURCE_CONTROL_B1
          DECISION:  RADAR_RESOURCE_CONTROL_B2
          ORIGINATING_REQUIREMENT:
          TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS.
     PATH :
        VALIDATION_POINT:  RADAR_COMMAND_OUTPUT_POINT
     END.


VALIDATION_PATH:  RADAR_WAVEFORM_PROPERTIES.
     REFERS TO:
          VALIDATION_POINT:  STARTING_POINT.
     CONSTRAINED BY:
          PERFORMANCE_REQUIREMENT:  ENERGY_PER_IMAGE
          PERFORMANCE_REQUIREMENT:  RADIATED_POWER.
     TRACED FROM:
          ORIGINATING_REQUIREMENT:
```

ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_1_B_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_1_B_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_A_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_A_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_B_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_B_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_C_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_C_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_D_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_D_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_E_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_2_E_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_A_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_A_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_B_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_B_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_C_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_C_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_D_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_D_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_E_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_E_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_F_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_F_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_G_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_3_G_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_A_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_A_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_B_FUNCTIONAL_REQUIREMENTS

```
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_4_B_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_A_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_A_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_B_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_B_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_C_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_C_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_D_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_PARAGRAPH_3_2_5_D_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_0_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_0_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_1_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_1_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_2_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SECTION_3_2_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_1_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_1_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_2_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_2_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_3_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_3_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_4_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_4_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_5_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_DPSPR_SUBSECTION_3_2_5_PERFORMANCE_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_RADAR_DPS_IFS_PARAGRAPH_3_2_9_FUNCTIONAL_REQUIREMENTS
ORIGINATING_REQUIREMENT:
TLS_RADAR_DPS_IFS_PARAGRAPH_3_2_9_PERFORMANCE_REQUIREMENTS.
```